Algoritma *Divide and Conquer*

(Bagian 4)

Bahan Kuliah IF2211 Strategi Algoritma

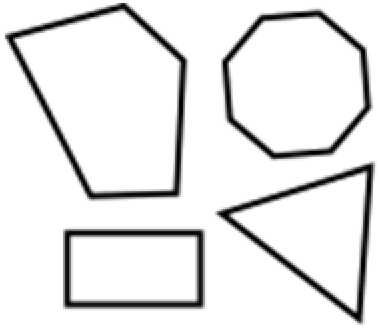
Oleh: Nur Ulfa Maulidevi & Rinaldi Munir

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika ITB
2025

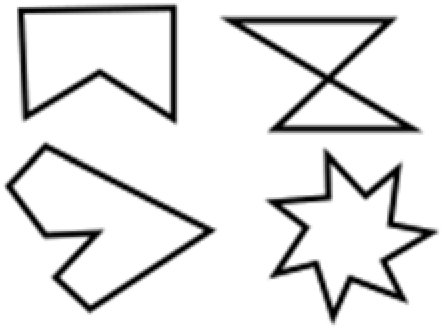
10. Convex Hull

- Salah satu hal penting dalam komputasi geometri adalah menentukan *convex hull* dari kumpulan titik.

- Himpunan titik pada bidang planar disebut *convex* jika untuk sembarang dua titik pada bidang tersebut (misal *p* dan *q*), seluruh segmen garis yang berakhir di *p* dan *q* berada pada himpunan tersebut.
- Contoh gambar 1 adalah poligon yang *convex*, sedangkan gambar 2 menunjukkan contoh yang *non-convex*.

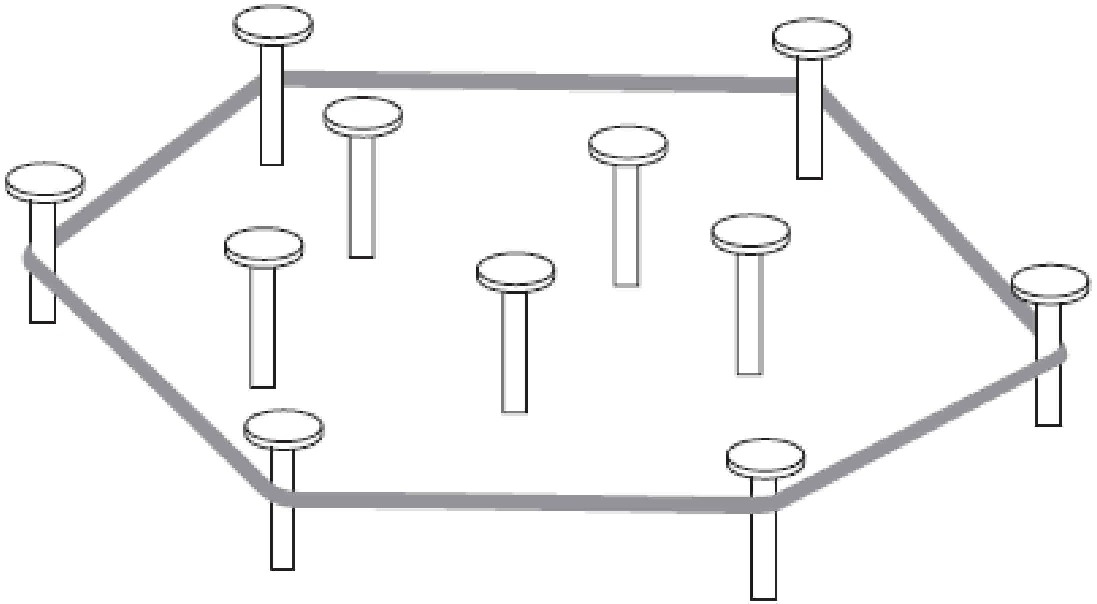


Gambar 1: convex



Gambar 2:non convex

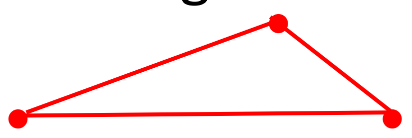
- *Convex hull* dari himpunan titik S adalah himpunan *convex* terkecil (*convex polygon*) yang mengandung S



- Untuk dua titik, maka *convex hull* berupa garis yang menghubungkan 2 titik tersebut.

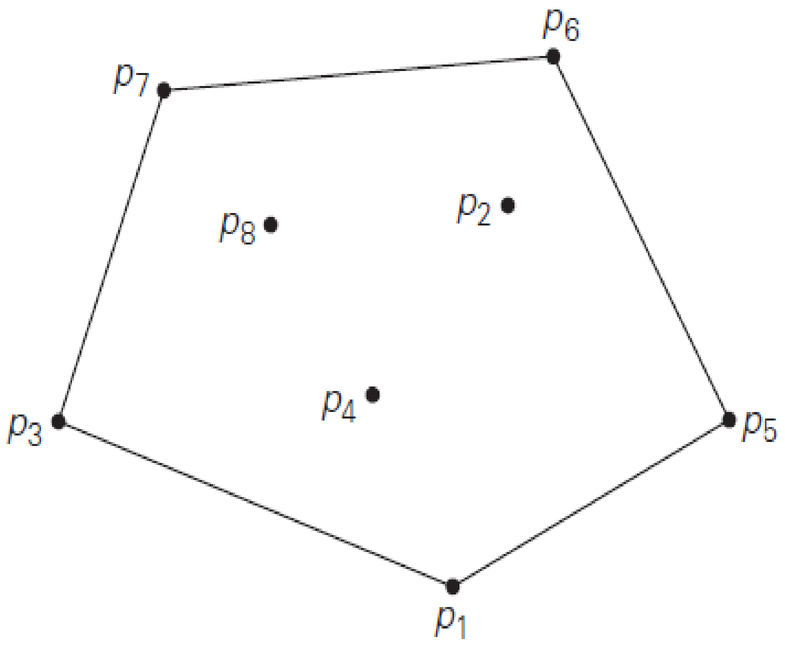
- Untuk tiga titik yang terletak pada satu garis, maka *convex hull* adalah sebuah garis yang menghubungkan dua titik terjauh.

- Sedangkan *convex hull* untuk tiga titik yang tidak terletak pada satu garis adalah sebuah segitiga yang menghubungkan ketiga titik tersebut.



- Untuk titik yang lebih banyak dan tidak terletak pada satu garis, maka *convex hull* berupa poligon *convex* dengan sisi berupa garis yang menghubungkan beberapa titik pada S.

- Contoh *convex hull* untuk delapan titik dapat dilihat pada gambar 3



Gambar 3 Convex Hull untuk delapan titik

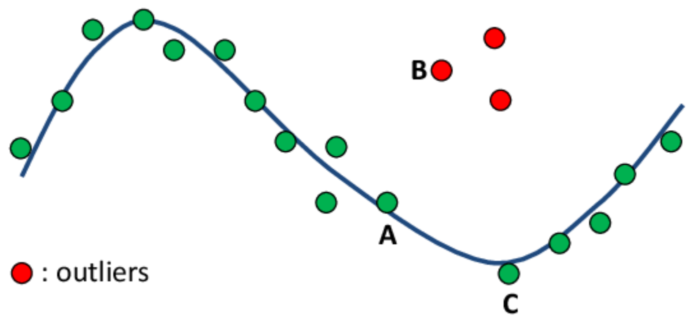
- Pemanfaatan dari *convex hull* ini cukup banyak.

- Pada animasi komputer, pemindahan suatu objek akan lebih mudah dengan memindahkan *convex hull* objek untuk *collision detection*.

- *Convex hull* juga dapat digunakan dalam persoalan optimasi, karena penentuan titik ekstrimnya dapat membatasi kandidat nilai optimal yang diperiksa.

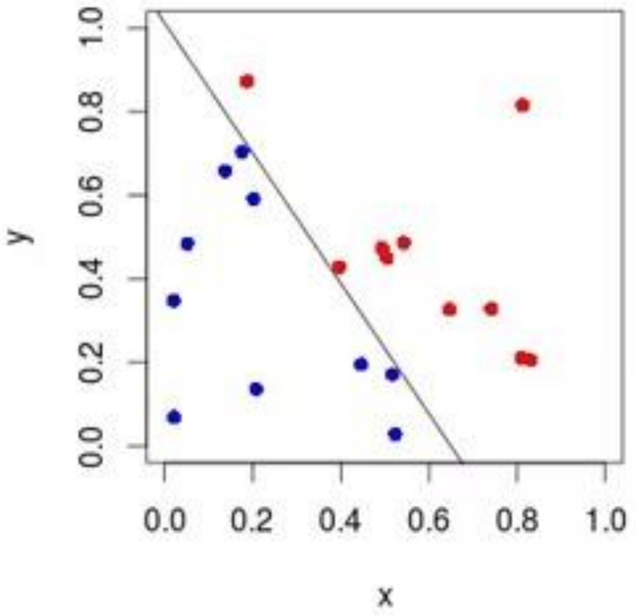
- Pada bidang statistik, *convex hull* juga dapat mendeteksi *outliers* pada kumpulan data.

  Outliers: titik-titik terluar

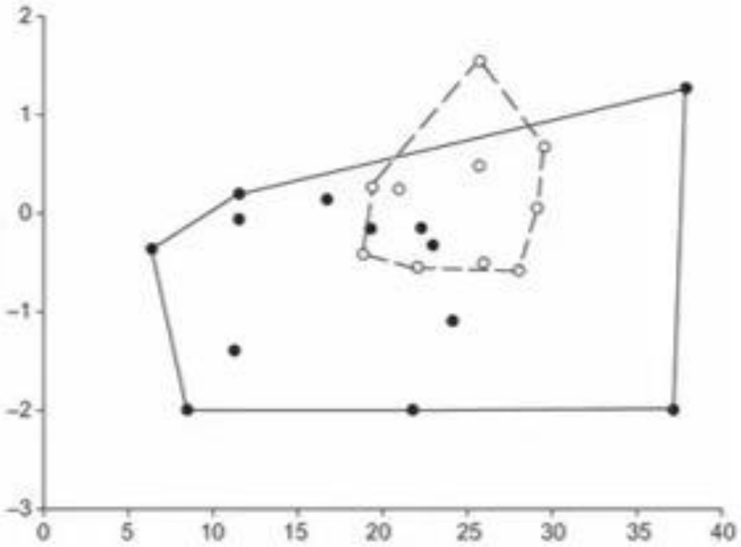


Pemanfaatan Convex Hull:
Tes Linearly Separable Dataset

Alternatif 1. Visualisasi data: analisis manual sulit utk dimensi tinggi.



Alternatif 2. Analisis convex-hull antar kelas tidak overlap → linearly separable dataset



https://www.quora.com/How-can-I-know-whether-my-data-is-linearly-separable

Convex Hull dengan Divide and Conquer

- Tujuan: menemukan kumpulan titik ‘terluar’ yang membentuk *convex hull*
- Ide dasar: menggunakan algoritma *Quicksort*

- S: himpunan titik sebanyak n, dengan $n>1$, yaitu titik $p_1(x_1, y_1)$ hingga $p_n(x_n, y_n)$ pada bidang kartesian dua dimensi

- Kumpulan titik diurutkan berdasarkan nilai absis yang menaik, dan jika ada nilai absis yang sama, maka diurutkan dengan nilai ordinat yang menaik

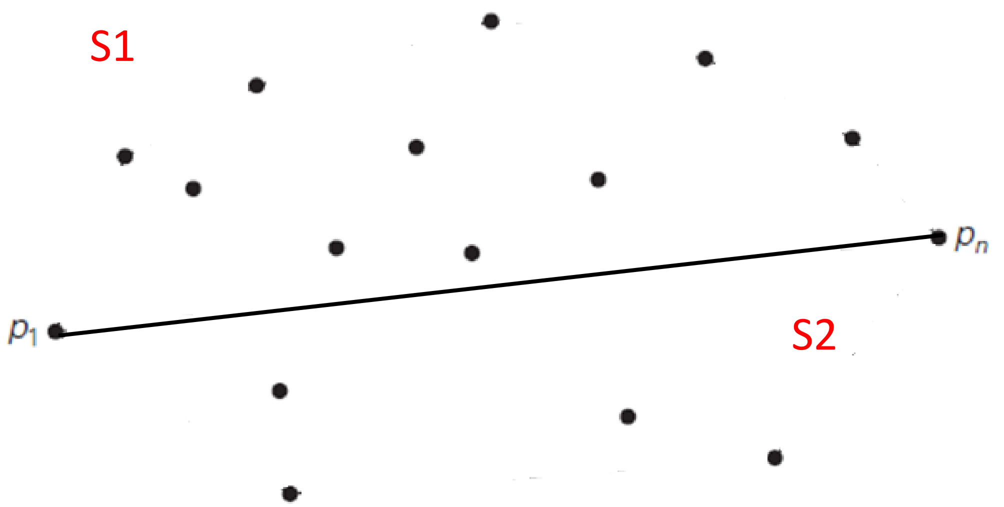
- p_1 dan p_n adalah dua titik ekstrim yang akan membentuk *convex hull* untuk kumpulan titik tersebut.

Ilustrasi:

Ide Divide and Conquer

1. Garis yang menghubungkan p_1 dan p_n $(p_1 p_n)$ membagi S menjadi dua bagian yaitu S1 (kumpulan titik di sebelah kiri atau atas garis $p_1 p_n$) dan S2 (kumpulan titik di sebelah kanan atau bawah garis $p_1 p_n$).

Ilustrasi:



Untuk memeriksa apakah sebuah titik berada di sebelah kiri (atau atas) suatu garis yang dibentuk dua titik, gunakan penentuan determinan:

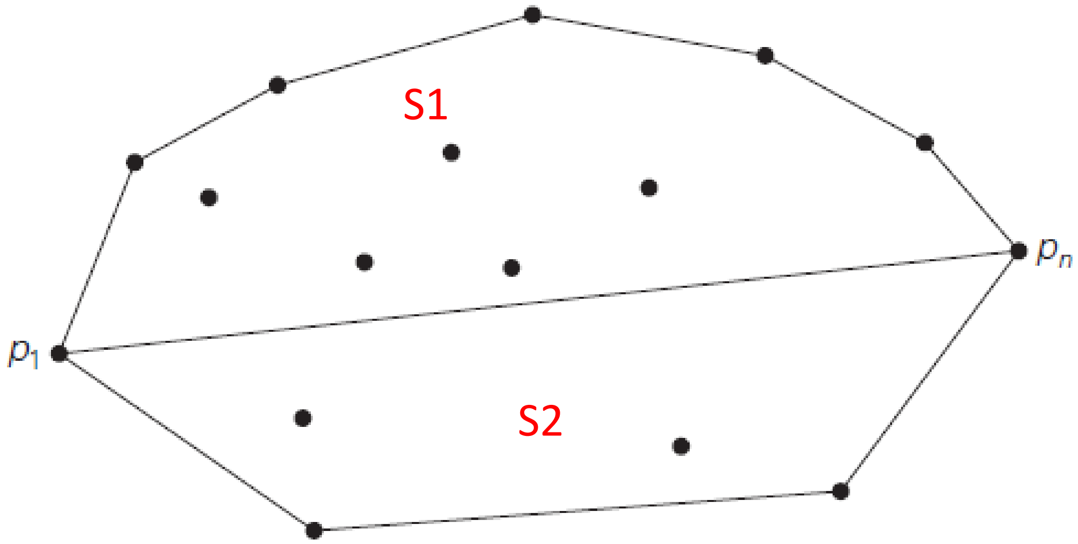
$$\begin{vmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{vmatrix} = x_1y_2 + x_3y_1 + x_2y_3 - x_3y_2 - x_2y_1 - x_1y_3$$

Titik (x3,y3) berada di sebelah kiri dari garis ((x1,y1),(x2,y2)) jika hasil determinan positif

2. Semua titik pada S yang berada pada garis p_1p_n (selain titik p_1 dan p_n) tidak mungkin membentuk *convex hull*, sehingga bisa diabaikan dari pemeriksaan

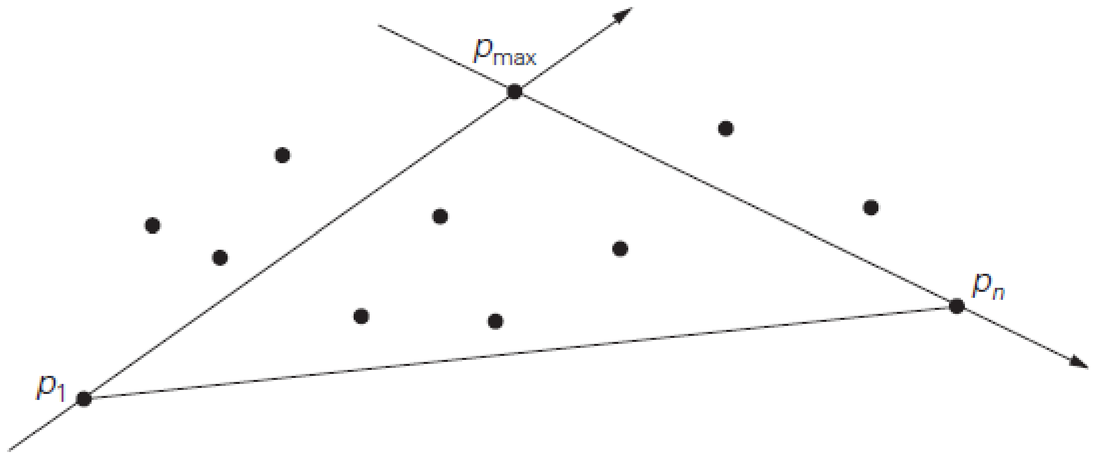
3. Kumpulan titik pada S1 bisa membentuk *convex hull* bagian atas, dan kumpulan titik pada S2 bisa membentuk *convex hull* bagian bawah → terapkan D & C

Ilustrasi:

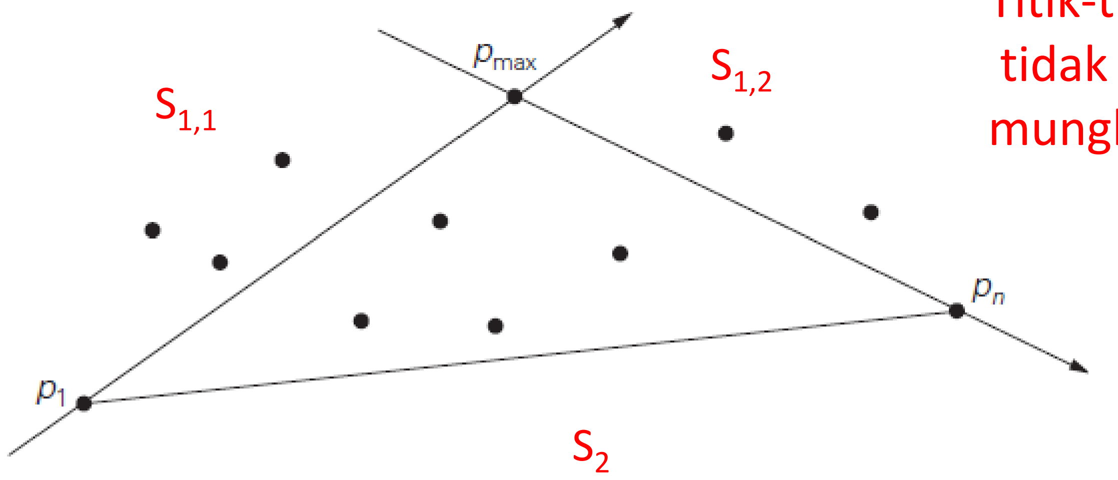


4. Untuk sebuah bagian (misal S1), terdapat dua kemungkinan:
   - Jika tidak ada titik lain selain S1, maka titik p_1 dan p_n menjadi pembentuk *convex hull* bagian S1
   - Jika S1 tidak kosong, pilih sebuah titik yang memiliki jarak terjauh dari garis p_1p_n (misal p_{max}). Jika terdapat beberapa titik dengan jarak yang sama, pilih sebuah titik yang memaksimalkan sudut $p_{max}\ p_1\ p_n$

   Semua titik yang berada di dalam daerah segitiga $p_{max}\ p_1\ p_n$ diabaikan untuk pemeriksaan lebih lanjut

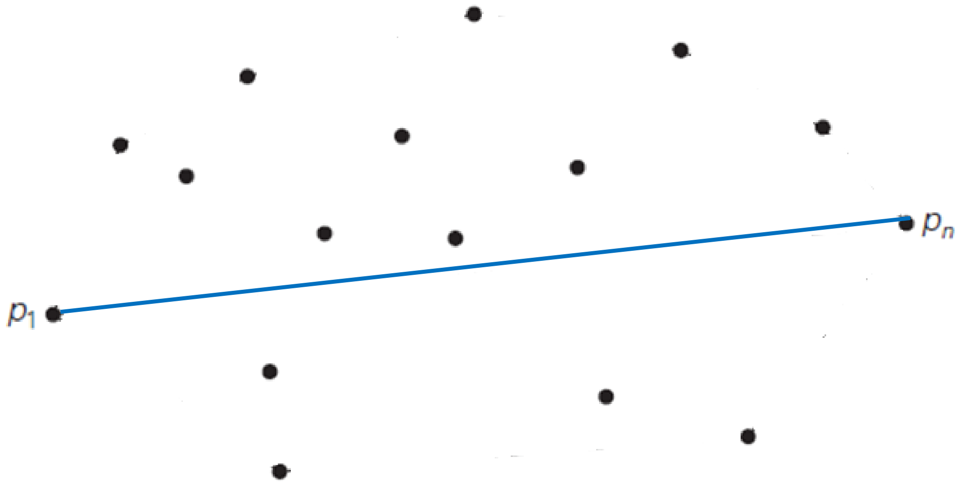


5. Tentukan kumpulan titik yang berada di sebelah kiri garis p_1p_{max} (menjadi bagian $S_{1,1}$), dan di sebelah kanan garis p_1p_{max}(menjadi bagian $S_{1,2}$)

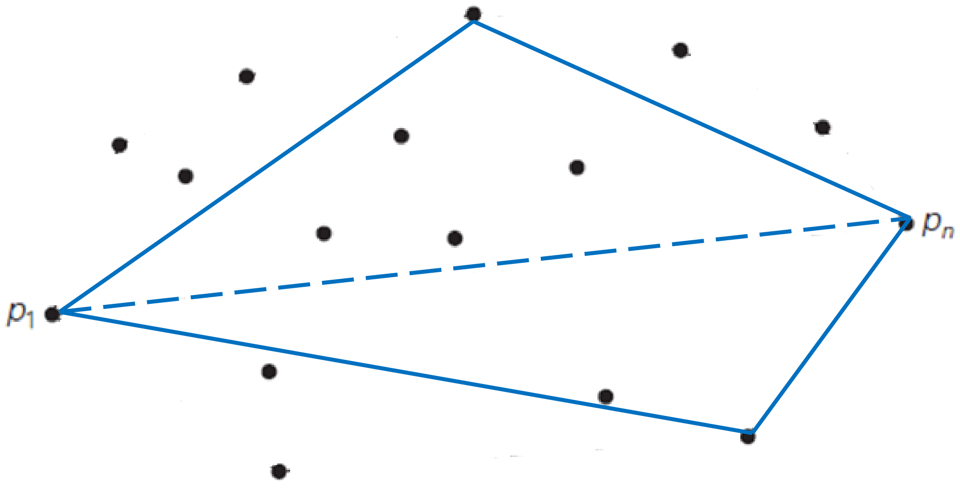


Titik-titik di dalam segitiga tidak diproses, karena tidak mungkin membentuk convex hull

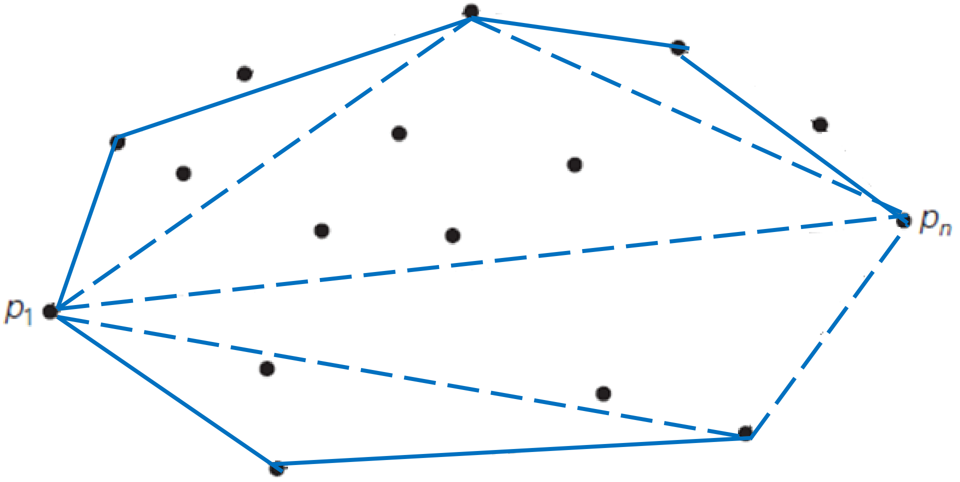
6. Lakukan hal yang sama (butir 4 dan 5) untuk bagian S2, hingga bagian ‘kiri’ dan ‘kanan’ kosong
7. Kembalikan pasangan titik yang dihasilkan



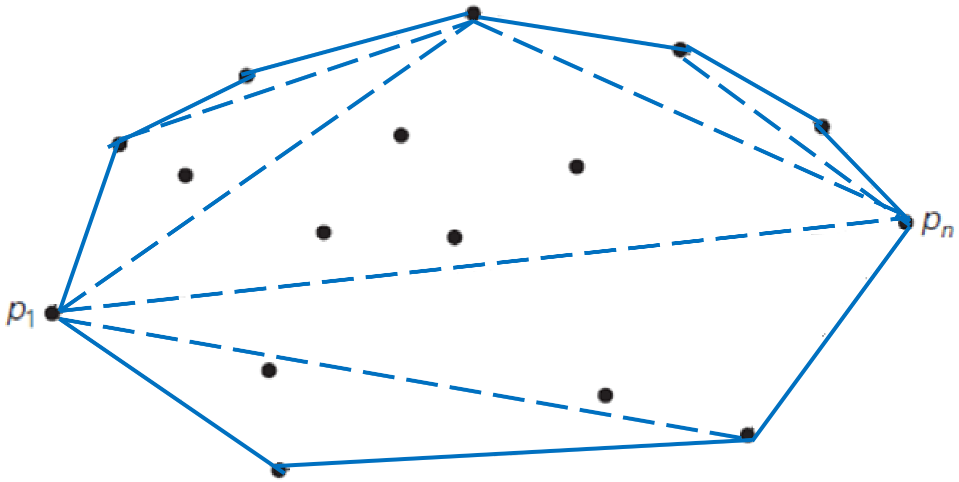
(i)



(ii)

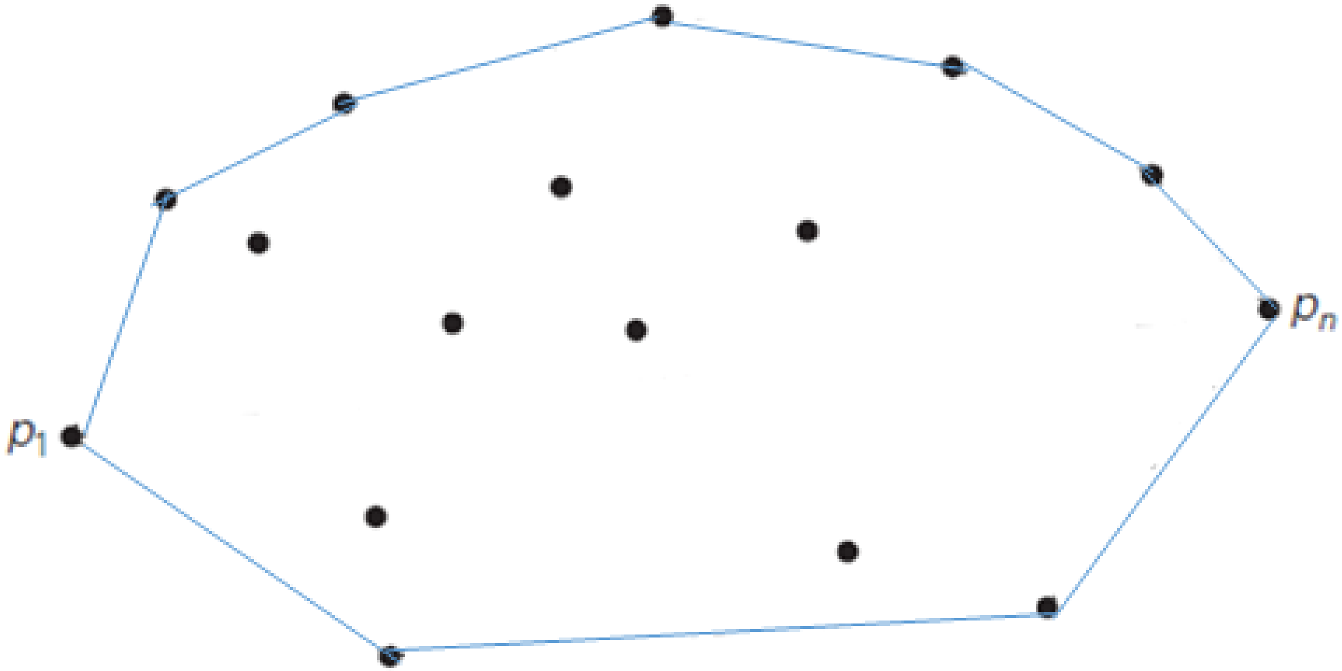


(iii)



(iv)

Hasil akhir:

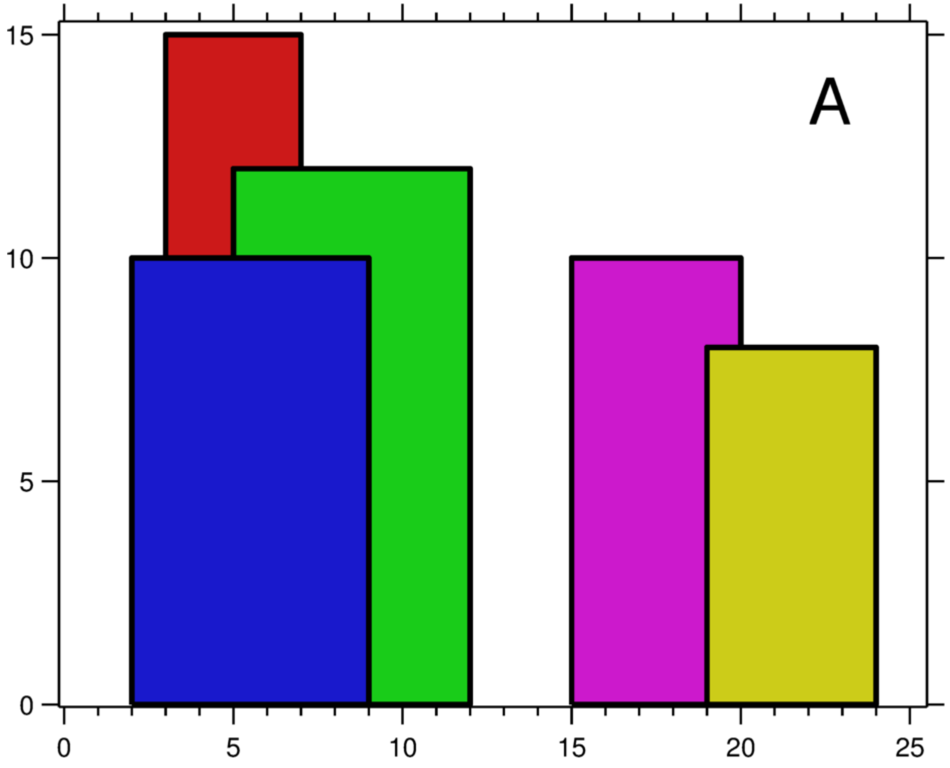


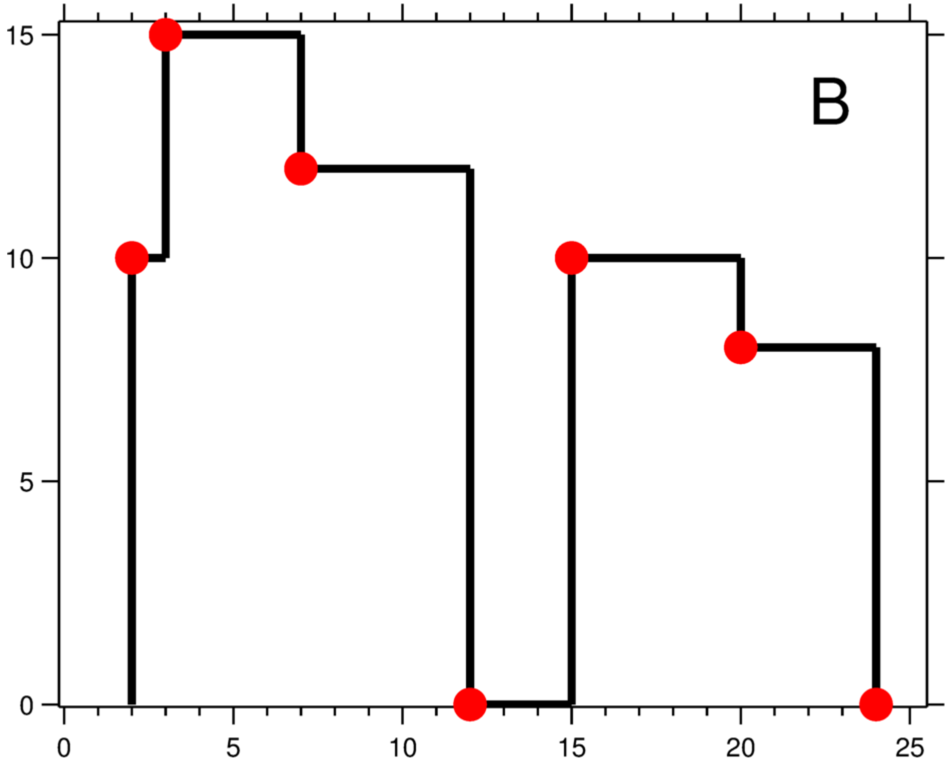
11. Skyline Problem*)

- Cakrawala kota (*skyline*) adalah kontur luar dari siluet yang dibentuk oleh semua bangunan di kota itu jika dilihat dari kejauhan.

*) Semua materi skyline problem di dalam ppt ini diambil dari https://www.learnbay.io/the-skyline-problem/

- Sekarang misalkan Anda diberikan lokasi dan ketinggian *n* buah bangunan seperti yang ditunjukkan pada gambar pemandangan kota (Gambar A), bagaimana menampilkan cakrawala yang dibentuk oleh bangunan-bangunan ini secara kolektif (Gambar B).





https://www.learnbay.io/the-skyline-problem/

- Informasi geometrik setiap bangunan diwakili oleh triplet [Li, Ri, Hi]

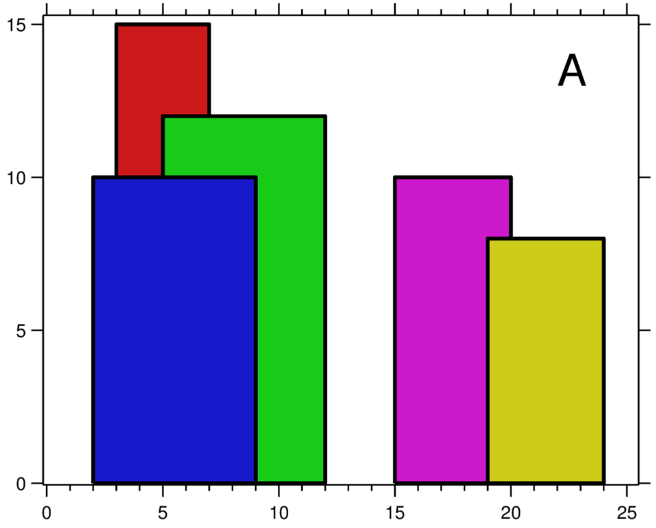
  Li : koordinat x tepi kiri bangunan ke-i,

  Ri : koordinat x tepi kanan bangunan ke-i,

  Hi : tinggi bangunan ke-i

  Li, Ri, Hi $\geq$ 0, dan Ri – Li > 0

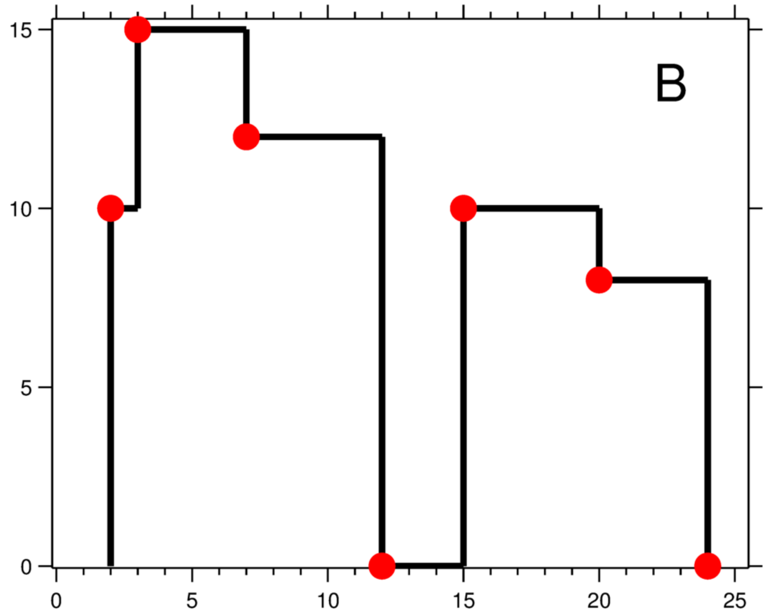
- Asumsi: semua bangunan berbentuk persegi panjang, permukaan tanah benar-benar datar pada ketinggian 0.



Gambar A: [[2 9 10], [3 7 15], [5 12 12], [15 20 10], [19 24 8]]

- Outputnya adalah daftar “titik kunci” (titik merah pada Gambar B) dalam format [[x1,y1], [x2, y2], [x3, y3], ...] yang secara unik mendefinisikan cakrawala.

- Titik kunci adalah titik akhir kiri dari segmen garis horizontal. Perhatikan bahwa titik kunci terakhir, di mana bangunan paling kanan berakhir, hanya digunakan untuk menandai berakhirnya kaki langit, dan selalu memiliki ketinggian nol. Juga, tanah di antara dua bangunan yang berdekatan harus dianggap sebagai bagian dari kontur kaki langit.



Gambar B: [[2 10], [3 15], [7 12], [12 0], [15 10], [20 8], [24, 0]]

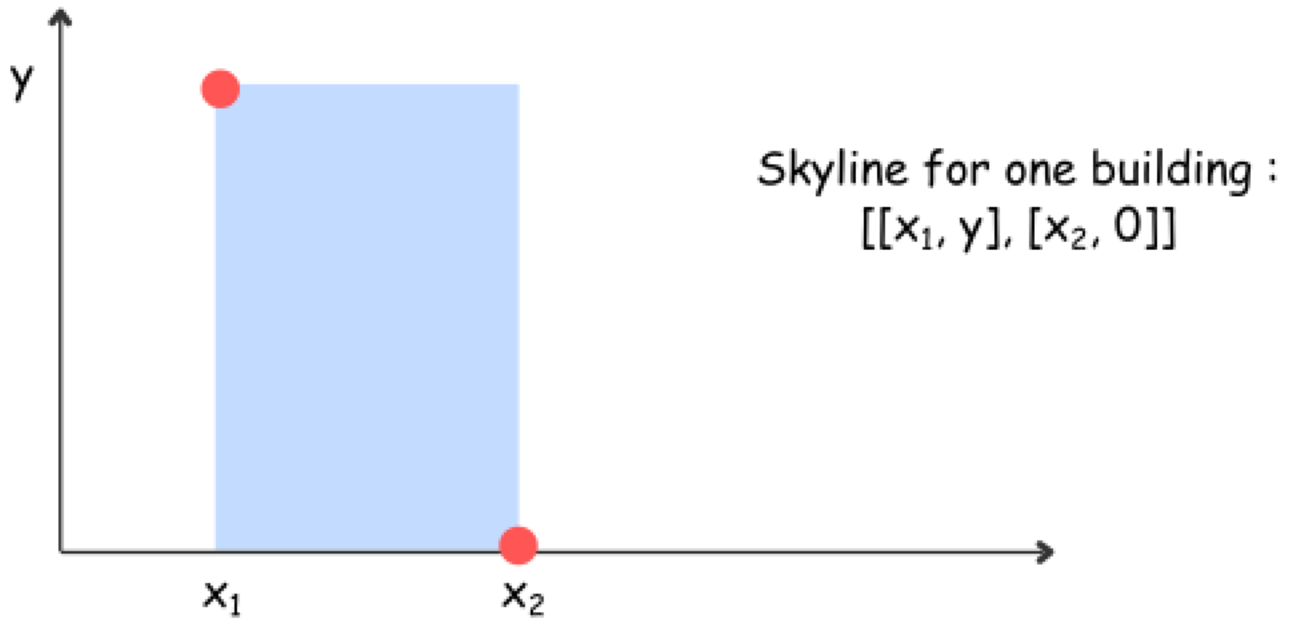
Penyelesaian *Skyline Problem* dengan algoritma *divide and conquer*

GetSkyline(n, *skyline)*

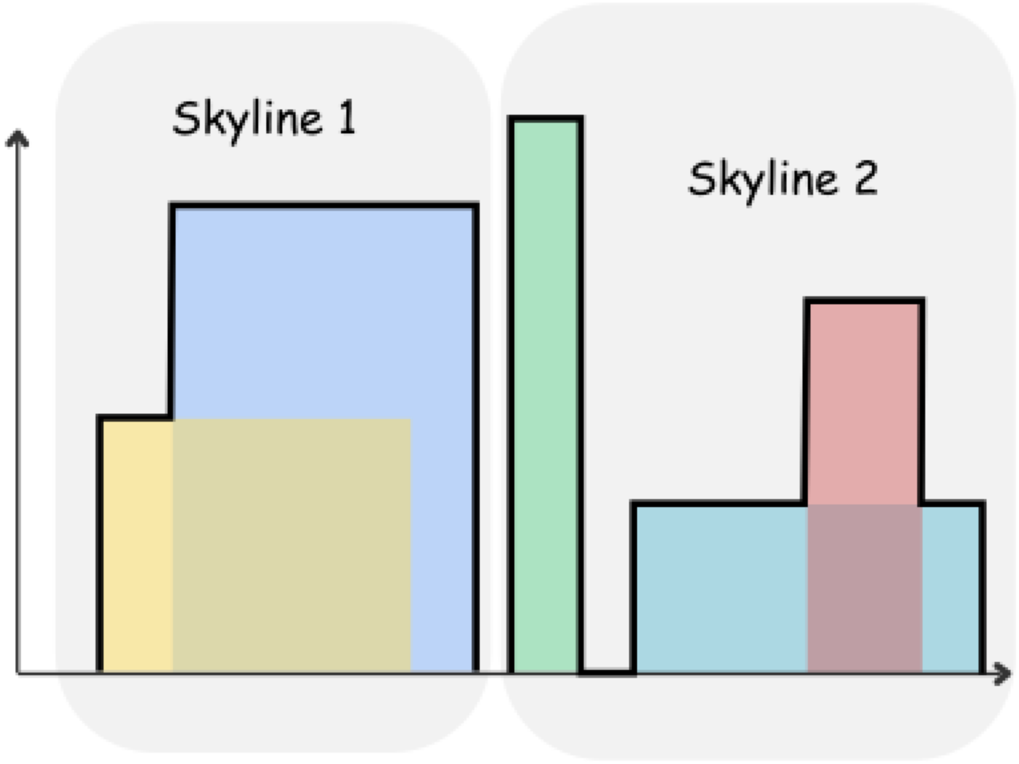
1. Jika n = 0, maka tidak ada *skyline* yang dihasilkan (skyline = [])
2. Jika n = 1, maka *skyline* yang dihasilkan adalah satu bangunan saja
3. Jika n > 1, maka
   - bagi n bangunan menjadi dua bagian, masing-masing n/2 bangunan
   - *GetSkyline*(n/2 bangunan yang pertama, *leftSkyline*)
   - *GetSkyline*(n/2 bangunan sisanya, *rightSkyline*)
   - Gabung(*leftSkyline*, *rightSkyline*)

- Kasus n = 0, tidak ada *skyline* yang dihasilkan

- Kasus n = 1, maka solusinya hanya satu buah *skyline* saja

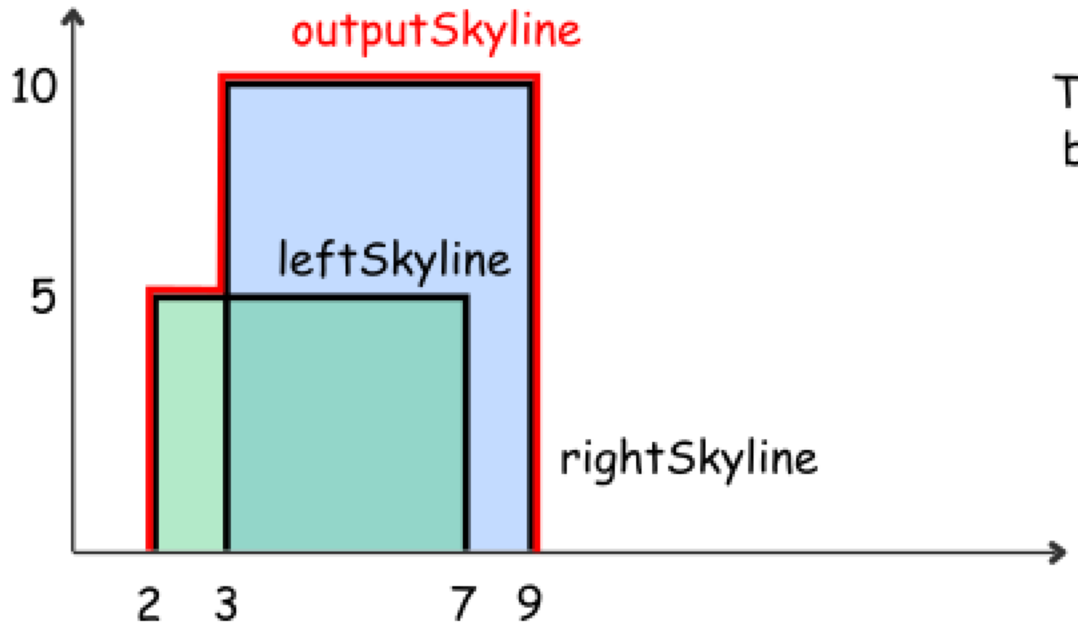


- Kasus n > 1 (tahap *conquer*):
  - bagi n bangunan menjadi dua bagian, masing-masing n/2 bangunan
  - tentukan *skyline* untuk masing-masing bagian



Split the buildings at each step
in two parts and
construct the skylines recursively
for each part

- Tahap *combine (merge)*:
  - gabungkan *skyline* masing-masing bagian
  - aturan logika: tinggi *skyline* output harus selalu yang maksimum di antara *skyline* kiri dan *skyline* kanan

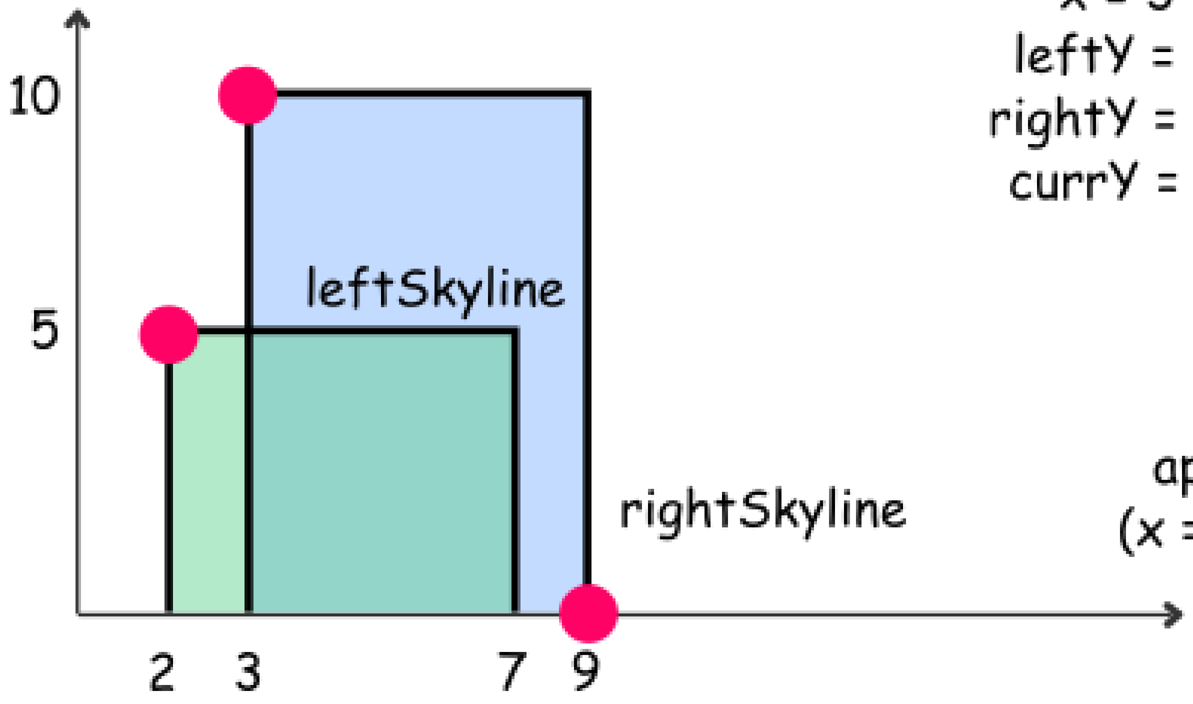


The output skyline is a maximum between left and right skylines

- Misalkan *pR* dan *pL* mencatat indeks elemen (bangunan) saat ini (*current*) pada kedua buah *skyline*, dan *leftY*, *rightY*, *currY* mencatat tinggi *left skyline*, *right skyline* dan *skyline* hasil gabungan.
- Idenya seperti *merge* di dalam *Mergesort*

mergeSkylines (left, right) :

- currY = leftY = rightY = 0
- While kedua *skyline* masih ada (pR < nR and pL < nL) :
  - Ambil elemen dengan koordinat x terkecil. Jika elemen tsb adalah elemen dari *left skyline*, *increment* pL dan perbarui leftY. Jika elemen tsb adalah elemen *right skyline*, *increment* pR dan perbarui rightY.
  - Hitung ketinggian terbesar pada titik saat ini: maxY = max(leftY, rightY).
  - Perbarui skyline output dengan titik (x, maxY), jika maxY tidak sama dengan currY
- Selagi masih terdapat bangunan di dalam left skyline (pL < nL), proses semuanya dengan aturan logika yang sama seperti yang dijelaskan di atas.
- Selagi masih terdapat bangunan di dalam left skyline (pR < nR), proses semuanya dengan aturan logika yang sama seperti yang dijelaskan di atas.
- Return *skyline* luaran (output).

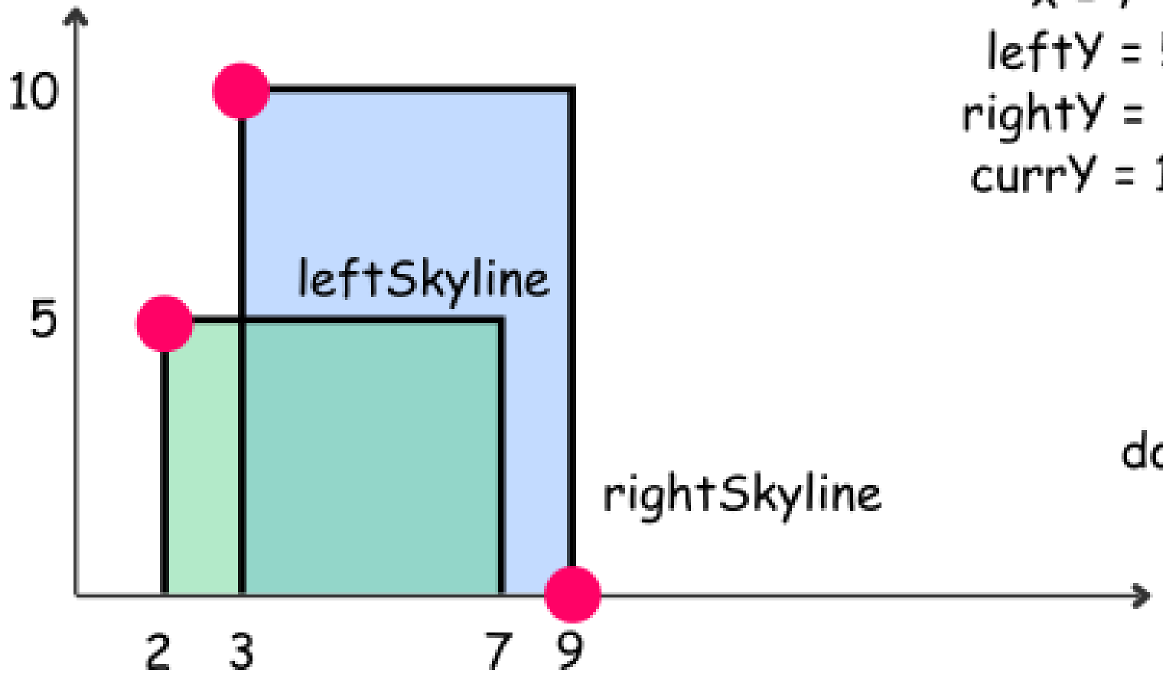


x = 3
leftY = 5
rightY = 10
currY = 5

skyline = [[2, 5]]

append the new point to skyline
(x = 3, y = max(leftY, rightY) = 10)

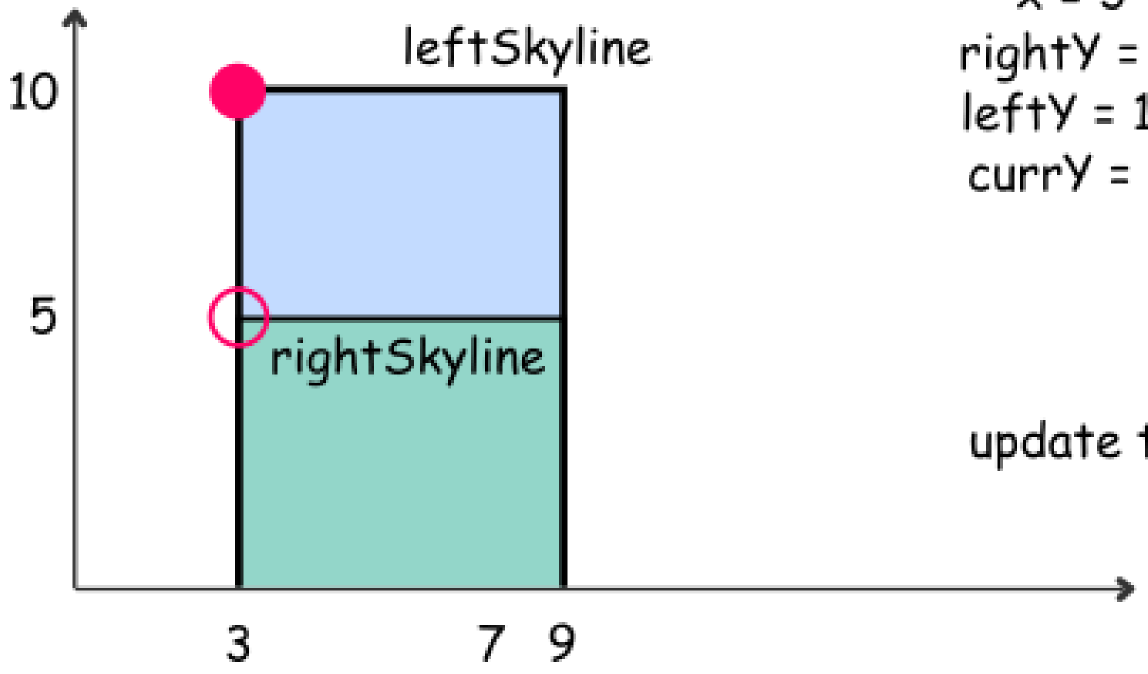
skyline = [[2, 5], [3,10]]



x = 7
leftY = 5
rightY = 10
currY = 10

skyline = [[2, 5], [3, 10]]

do NOT update skyline because
currY = max(leftY, rightY)



x = 3
rightY = 5
leftY = 10
currY = 5

skyline = [[3, 5]]

update the last point [3, 5] -> [3, 10] of the skyline
because the new x is the same
(= the vertical skyline change)

skyline = [[3,10]]

PYTHON IMPLEMENTATION

Sumber kode: https://www.learnbay.io/the-skyline-problem/

```python
class Solution:
    def getSkyline(self, buildings: 'List[List[int]]') -> 'List[List[int]]':
        """
        Divide-and-conquer algorithm to solve skyline problem,
        which is similar with the merge sort algorithm.
        """
        n = len(buildings)
        # The base cases
        if n == 0:
            return []
        if n == 1:
            x_start, x_end, y = buildings[0]
            return [[x_start, y], [x_end, 0]]

        # If there is more than one building,
        # recursively divide the input into two subproblems.
        left_skyline = self.getSkyline(buildings[: n // 2])
        right_skyline = self.getSkyline(buildings[n // 2 :])
```

```python
        # Merge the results of subproblem together.
        return self.merge_skylines(left_skyline, right_skyline)

    def merge_skylines(self, left, right):
        """
        Merge two skylines together.
        """
        def update_output(x, y):
            """
            Update the final output with the new element.
            """
            # if skyline change is not vertical -
            # add the new point
            if not output or output[-1][0] != x:
                output.append([x, y])
            # if skyline change is vertical -
            # update the last point
            else:
                output[-1][1] = y

        def append_skyline(p, lst, n, y, curr_y):
            """
```

```
        """
        while p < n:
            x, y = lst[p]
            p += 1
            if curr_y != y:
                update_output(x, y)
                curr_y = y

    n_l, n_r = len(left), len(right)
    p_l = p_r = 0
    curr_y  = left_y = right_y = 0
    output = []

    # while we're in the region where both skylines are present
    while p_l < n_l and p_r < n_r:
        point_l, point_r = left[p_l], right[p_r]
        # pick up the smallest x
        if point_l[0] < point_r[0]:
            x, left_y = point_l
            p_l += 1
```

```python
        else:
            x, right_y = point_r
            p_r += 1
        # max height (i.e. y) between both skylines
        max_y = max(left_y, right_y)
        # if there is a skyline change
        if curr_y != max_y:
            update_output(x, max_y)
            curr_y = max_y

    # there is only left skyline
    append_skyline(p_l, left, n_l, left_y, curr_y)

    # there is only right skyline
    append_skyline(p_r, right, n_r, right_y, curr_y)

    return output
```

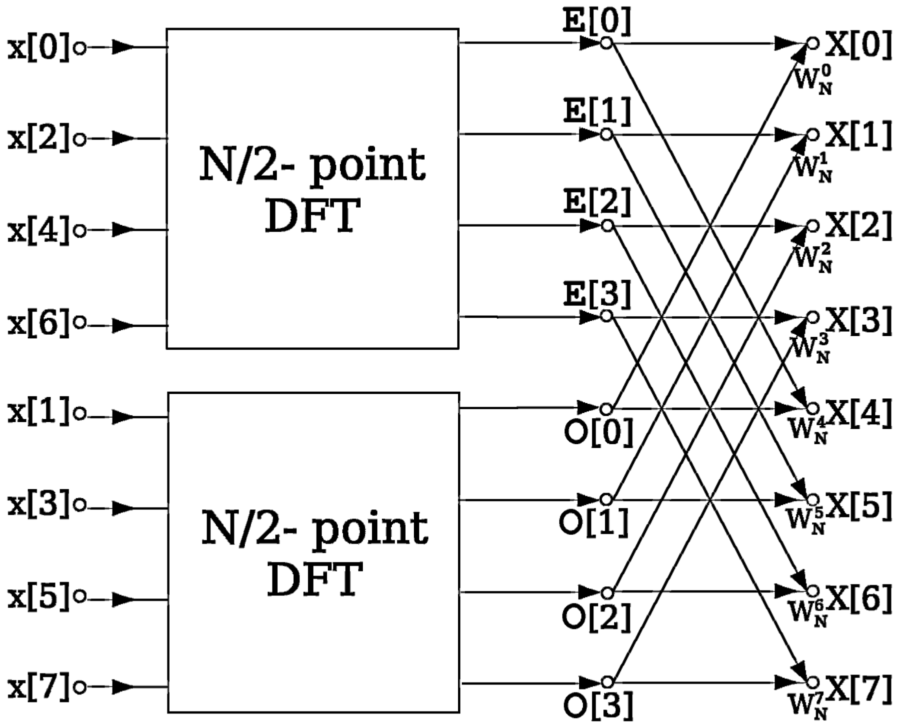
Contoh algoritma lain yang menggunakan divide and conquer:

1. FFT - Fast Fourier Transform (Algoritma Cooley-Tukey)

$$X(k) = \sum_{n=0}^{N-1} x(n) W_N^{kn} \qquad 0 \le k \le N-1$$

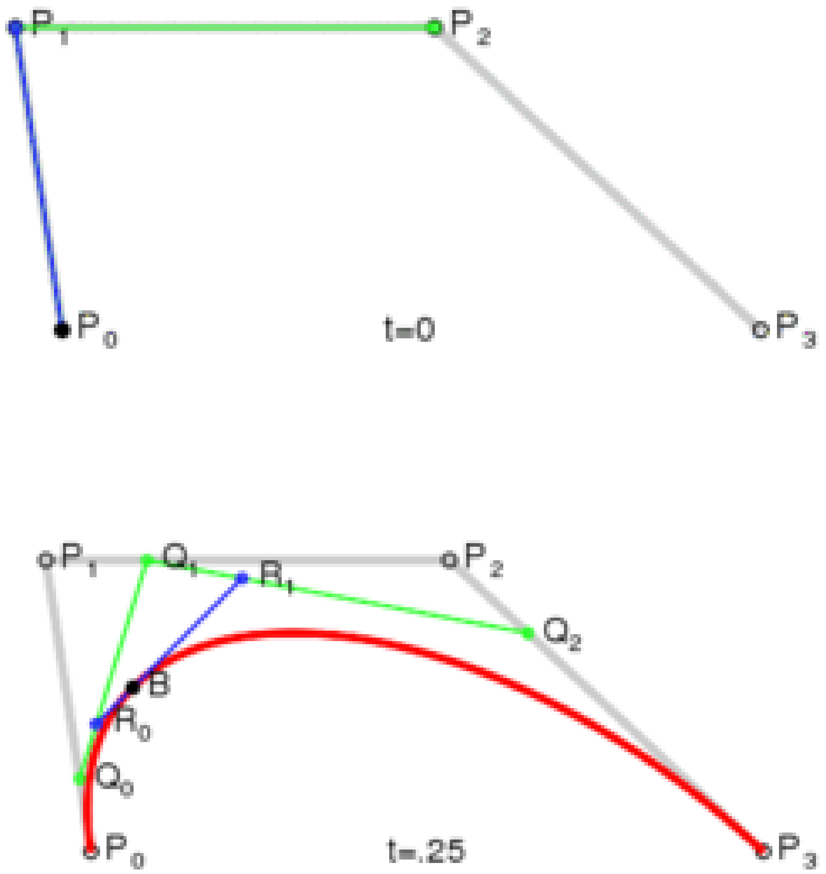
$$x(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k) W_N^{-nk} \qquad 0 \le n \le N-1$$

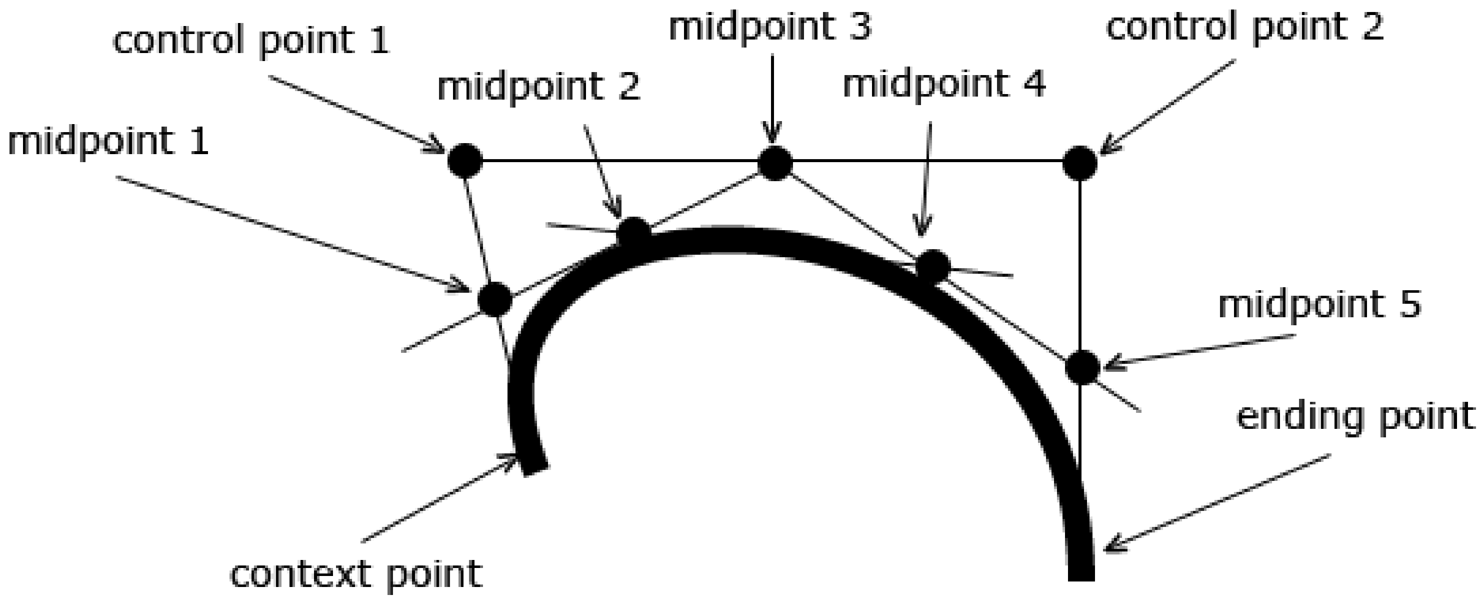
$$W_N = e^{-j\frac{2\pi}{N}}$$



2. Kurva Bezier di dalam grafika computer (*computer graphics*)

→ Metode untuk menggambarkan kurva mulus (smooth)





Soal Latihan 2 (UTS 2019) – Optimal Subsequence yang kontigu

Diberikan sebuah larik (array) integer $a_1, a_2, \ldots, a_n$. Anda diminta menemukan *sub-sequence* yang kontigu (berderetan) dari larik tersebut yang memiliki nilai maksimum. Sebagai contoh:

[–2 , 11, –4 , 13, –5 , 2, –1, 3]

memiliki nilai maksimum *sub-sequence* kontigu = 20, yaitu [11, –4 , 13].

Penyelesaian dengan algoritma brute force (lihat materi Algoritma *Brute Force* Bagian 2) memiliki kompleksitas algoritma $O(n^2)$. Jika diselesaikan dengan algoritma *divide and conquer* bagaimana caranya (langkah-langkahnya, bukan *pseudo-code*)? Jelaskan jawaban anda dengan mengambil contoh larik di atas. Berapa kompleksitas waktu asimptotiknya?

(Petunjuk: gunakan gagasan seperti pada masalah mencari sepasang jarak titik terdekat/ *The Closest Point Pair Problem*).

Penyelesaian:

Algoritma *divide and conquer*:

if n = 1, maka

nilai maksimum = elemen tersebut

else

- bagi menjadi dua upa-larik.
- Nilai maksimum *sub-sequence* yang kontigu dapat terjadi pada salah satu dari tiga kasus berikut:
  a) Case 1: semua elemen *sub-sequence* yang berjumlah maksimum terdapat pada upa-larik kiri.
  b) Case 2: semua elemen *sub-sequence* yang berjumlah maksimum terdapat pada upa-larik kanan
  c) Case 3: elemen *sub-sequence* yang berjumlah maksimum dimulai pada upalarik kiri dan berakhir pada upa-larik kanan

  Ambil nilai terbesar dari langkah (a), (b), dan (c)

Detil setiap kasus:

- Case 1: Hitung secara rekursif nilai maksimum *sub-sequence* yang seluruhnya terdapat pada upa-larik kiri.

- Case 2: Hitung secara rekursif nilai maksimum *sub-sequence* yang seluruhnya terdapat pada upa-larik kanan

- Case 3: Hitung nilai maksimum *sub-sequence* yang berawal pada upa-larik kiri dan berakhir pada upa-larik kanan. Caranya adalah:

    - Cari jumlah maksimum upalarik mulai dari elemen tengah ke kiri

    - Cari jumlah maksimum upalarik mulai dari elemen tengah+1 ke kanan

    - Kombinasikan keduanya dan jumlahkan hasilnya

Contoh: [–2 , 11, –4 , 13, –5 , 2, –1, 3]

[–2 , 11, –4 , 13]   [–5 , 2, –1, 3]

Kasus 1: sub-array kontigu maksimum di bagian kiri: [11, –4 , 13] → 11 – 4 + 13 = 20

Kasus 2: sub-array kontigu maksimum di bagian kanan: [2, –1 , 3] → 2 – 1 + 3 = 4

Kasus 3: sub-array kontigu max di antara dua bagian: [11, –4 , 13, -5, 2, -1, 3] → 19

Max(20, 4, 10) = 20 → [11, –4 , 13]

Rincian:

–2 11 –4 13 –5 2 –1 3

–2 11 –4 13 | –5 2 –1 3

–2 11 | –4 13 | –5 2 | –1 3

–2 | 11 | –4 | 13 | –5 | 2 | –1 | 3

–2 | 11 | –4 | 13 | –5 | 2 | –1 | 3

m=-2 m =11 m =-4 m = 13 m =-5 m = 2 m = -1 m = 3

divide sekaligus conquer

Ket: m = nilai max sub-array kontigu

Combine:

–2 11	–4 13	–5 2	–1 3
m1 = -2→[-2]	m1 = -4→[-4]	m1 = -5→[-5]	m1 = -1→[-1]
m2 = 11→[11]	m2 = 13→[13]	m2 = 2→[2]	m2 = 3→[3]
m3 = 9→[-2, 11]	m3 = 9→[-4, 13]	m3 = -3→[-5,2]	m3 = 2→[-1, 3]
max = 11→[11]	**max = 13→[13]**	**max = 2→[2]**	**max = 3→[3]**

–2 11 –4 13

m1 = 11, m2 = 13, m3 = 20→[11,-4,13]

max = 20 → [11, -4, 13]

–5 2 –1 3

m1 = 2, m2 = 3, m3 = 4→[2,-1,3]

max = 4 → [2, -1, 3]

–2 11 –4 13 –5 2 –1 3

m1 = 20 → [11, -4, 13]
m2 = 4 → [2, -1, 3]
m3 = 19 → [11, –4 , 13, -5, 2, -1, 3]
max = 20 → [11, -4, 13]

Kompleksitas waktu algoritma:

T(*n*) adalah jumlah operasi penjumlahan

Pada tahap *conquer* terdapat dua pemanggilan rekursif, masing-masing untuk *n*/2 elemen larik.

Operasi penjumlahan untuk upalarik sepanjang maks *n* elemen = *cn*

Untuk *n* = 1, operasi penjumlahan = 0, secara umum = *a*.

$$T(n) = \begin{cases} a, & n = 1 \\ 2T(n/2) + cn, & n > 1 \end{cases}$$

Menurut Teorema Master, $T(n) = O(n \log n)$

Latihan Soal

1. (UTS 2020)

a. MergeSort merupakan salah satu teknik pengurutan dengan Divide and Conquer yang memiliki kompleksitas algoritma $O(n \log_2 n)$. Jika diberikan suatu array sembarang berukuran $n=2^k$, jelaskanlah berapa kali pemanggilan rekursif MergeSort dan Merge dilakukan dalam pengurutan array tersebut. Pemanggilan pertama MergeSort(A,1,n) masuk dalam perhitungan. (5)

b. Lakukanlah proses divide atau partisi pada QuickSort untuk array karakter 'MARET' dengan menggunakan pivot elemen tengah sehingga array terurut menaik. Indeks array dimulai dari 1. Pada setiap langkah perjelas posisi yang ditukar, dan posisi pada tabel di mana partisi dilakukan. (7.5)

c. Diberikan kumpulan titik [{1, 1}, {2, 2}, {4, 4},{0, 0}, {1, 2}, {3, 1}, {3, 3}], lakukanlah proses QuickHull yang menggunakan strategi Divide and Conquer untuk mendapatkan kumpulan titik yang membentuk convex hull. Tidak perlu melakukan perhitungan jarak antara dua titik, boleh menggunakan estimasi untuk prosesnya. (7.5)

2. (UTS 2016)

(TEOREMA MASTER) Aplikasikan Teorema Master untuk menentukan notasi *Big-Oh* dari relasi rekurens berikut. Jika tidak bisa diaplikasikan, tuliskan "tidak bisa diterapkan".

(a) $T(n) = 4T(n/2) + n^2\sqrt{n}$

(b) $T(n) = 7T(n/3) + n^3$

(c) $T(n) = 2T(n/3) + 1$

(d) $T(n) = 9T(n/3) + n/(\log n)$ (Nilai: 10)

3. (UTS 2015)

(Teorema Master) Gunakan teorema Master untuk menentukan notasi asymptotic untuk $T(n)$ berikut:

(a) $T(\mathrm{n}) = 8T(n/2) + n^2$

(b) $T(n) = 2T(n/2) + n^2$

(c) $T(n) = 4T(n/2) + n^2$

(d) $T(n) = 2T\left(\sqrt{n}\right) + \log n^{1.5}$ (Petunjuk: misalkan $n = 2^m$, lalu definisikan $S(m) = T(2^m)$. Jelaslah bahwa solusi $S(m)$ juga solusi $T(n)$)

(Nilai = 3 + 3 + 3 + 6)

4. (UTS 2015)

(***Brute Force, Divide and Conquer***) Seorang ahli biologi sedang meneliti n sampel DNA. Menemukan kode DNA eksak untuk tiap sampel memakan waktu yang lama. Namun, semua ahli biologi ingin mengetahui apakah terdapat paling sedikit setengah dari sampel berasal dari hewan yang sama. Dia ingin memperoleh metode yang dengan cepat menentukan apakah sembarang pasangan sampel DNA berasal dari hewan yang sama. Dengan mengasumikan n adalah perpangkatan dari dua, maka:

(a) Bagaimana algoritma *brute force* untuk menentukan apakah paling sedikit setengah dari n sampel DNA tersebut berasal dari hewan yang sama. Deskripsikan algoritmanya (bukan *pseudo-code*) dan perkirakan kompleksitas algoritmanya dalam notasi O-besar.

(b) Rancanglah algoritma *divide and conquer* yang menentukan apakah paling sedikit dari n sampel DNA berasal dari hewan yang sama. Deskripsikan algoritmanya masing-masing untuk bagian basis dan bagian rekurens (bukan *pseudo-code*), hitung kompleksitas waktunya dalam $T(n)$ yang berbentuk rekursif , lalu perkirakan kompleksitas algoritmanya dalam notasi O-besar (menggunakan Teorema Master).

(Nilai = 10 + 10)

5. Inversion Problem

(*Inversion problem*) *Netflix* menggunakan sistem rekomendasi untuk merekomendasikan film yang anda sukai. *Netflix* mencoba mencocokkan film kesukaanmu dengan film lainnya. Sistem rekomendasi tersebut adalah sbb: Misalkan kamu me-rangking *n* buah film. Selanjutnya, *Netflix* memeriksa basisdatanya untuk mencari orang dengan kesukaan film yang mirip. Ukuran kemiripan yang digunakan adalah jumlah inversi antara kedua rangking. Misalkan ranking dari orang tersebut adalah 1, 2, 3, …, n, sedangkan rangking dari kamu adalah a_1, a_2, …, a_n. Film i dan film j disebut inversi jika $i < j$ tetapi $a_i > a_j$. Contoh untuk film A, B, C, D, dan E:

Film	A	B	C	D	E
Ranking saya	1	2	3	4	5
Ranking X	1	3	4	2	5

Inversi: (3, 2) dan (4, 2)

Film	A	B	C	D	E
Ranking saya	1	2	3	4	5
Ranking Y	1	2	4	3	5

Inversi (4, 3)

Karena jumlah inversi dengan Y lebih sedkit daripada X, maka kesukaan saya lebih mirip dengan Y.

Jika diselesaikan dengan algoritma *Divide and Conquer*, bagaimana langkah-langkahnya? Jelaskan dengan contoh senarai delapan elemen! Berapa jumlah perbandingan elemen yang dibutuhkan dan berapa kompleksitas algoritma dalam notasi *Big-Oh*? Apakah kompleksitas algoritmanya lebih baik dari *brute force*? **(10)**

Jawaban Soal Inversion Problem

Oleh: Rinaldi M

(*Inversion problem*) *Netflix* menggunakan sistem rekomendasi untuk merekomendasikan film yang anda sukai. *Netflix* mencoba mencocokkan film kesukaanmu dengan film lainnya. Sistem rekomendasi tersebut adalah sbb: Misalkan kamu me-rangking n buah film. Selanjutnya, *Netflix* memeriksa basisdatanya untuk mencari orang dengan kesukaan film yang mirip. Ukuran kemiripan yang digunakan adalah jumlah inversi antara kedua rangking. Misalkan ranking dari orang tersebut adalah 1, 2, 3, ..., n, sedangkan rangking dari kamu adalah a_1, a_2, ..., a_n. Film i dan film j disebut inversi jika $i < j$ tetapi $a_i > a_j$. Contoh untuk film A, B, C, D, dan E:

Film	A	B	C	D	E
Ranking saya	1	2	3	4	5
Ranking X	1	3	4	2	5

Inversi: (3, 2) dan (4, 2)

Film	A	B	C	D	E
Ranking saya	1	2	3	4	5
Ranking Y	1	2	4	3	5

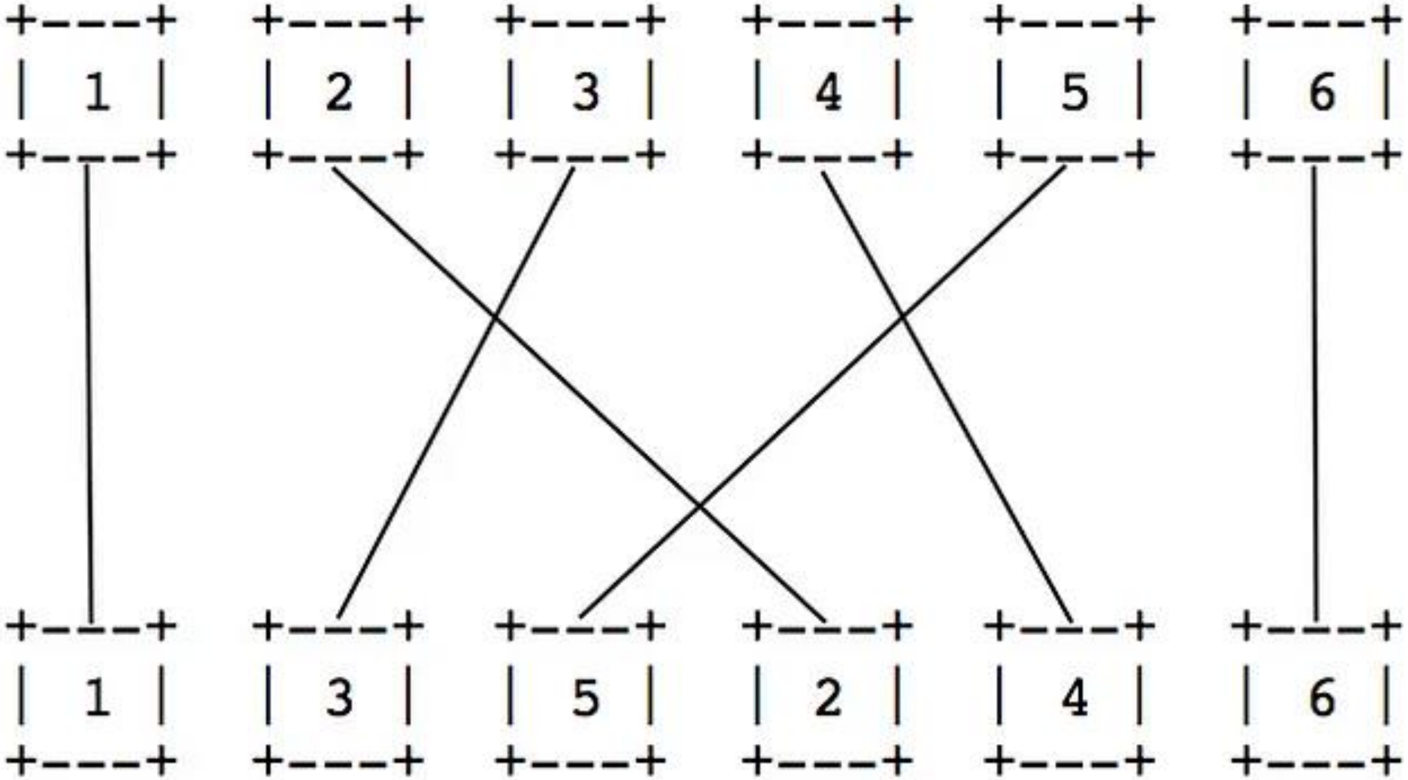
Inversi (4, 3)

Karena jumlah inversi dengan Y lebih sedkit daripada X, maka kesukaan saya lebih mirip dengan Y.

Jika diselesaikan dengan algoritma *Divide and Conquer*, bagaimana langkah-langkahnya? Jelaskan dengan contoh senarai delapan elemen! Berapa jumlah perbandingan elemen yang dibutuhkan dan berapa kompleksitas algoritma dalam notasi *Big-Oh*? Apakah kompleksitas algoritmanya lebih baik dari *brute force*? **(10)**

Perangkingan saya:

Perangkingan orang lain:



Ada 3 buah inversi: (3, 2), (5, 2), dan (5, 4)

Jawaban:

Divide: bagi larik a menjadi dua bagian, masing-masing berukuran n/2 elemen

Conquer: hitung inversi pada masing-masing bagian.

Combine: hitung inversi di mana a_i dan a_j pada masing-masing bagian, lalu jumlahkan inversi ketiganya

```
function HitungInversi(A, n)
if  n = 1
      return 0
else
     k ← bagidua larik pada posisi n/2, masing-masing A1 dan A2
     x ← HitungInversi(A1, n/2)
     y ← HitungInversi(A2, n/2)
     z ← hitung inversi elemen di A1 dan di A2(A, n)
     return x + y + z
endif
```

Contoh:

1	5	4	8	10	2	6	9	12	11	3	7

1	5	4	8	10	2

6	9	12	11	3	7

5 inversi: (5, 4), (5, 2),
(4, 2), (8, 2), (10,2)

8 inversi: (6, 3), (9, 3), (9,7)
(12, 11), (12, 3), (12,7), (11, 3), (11, 7),

9 inversi biru-hijau: (5, 3), (4, 3), (8, 6), (8, 3), (8, 7)
(10, 6), (10, 9), (10, 3), (10, 7)

Total = 5 + 8 + 9 = 22 buah inversi

Combine: hitung inversi biru-hijau

- Asumsikan setiap bagian larik terurut.
- Hitung inversi a_i dan a_j pada masing-masing bagian.
- Gabung (*merge*) dua bagian larik yang terurut menjadi sebuah larik terurut

3	7	10	14	18	19

2	11	16	17	23	25
6	3	2	2	0	0

13 inversi biru-hijau:
(3, 2), (7, 2), (10, 2), (14, 2), (18, 2), (19, 2), (6 inversi terhadap 2)
(14, 11), (18, 11), (19, 11), (3 inversi terhadap 11)
(18, 16), (19, 16), (2 inversi terhadap 16)
(18, 17), (19, 17), (2 inversi terhadap 17)

2	3	7	10	11	14	16	17	18	19	23	25

Merge: O(n)

Kompleksitas waktu: $T(n) = \begin{cases} a & , n = 1 \\ 2T\left(\frac{n}{2}\right) + cn & , n > 1 \end{cases}$

Jika diselesaikan dengan Teorema Master,
Maka T(n) = 2T(n/2) + cn adalah O(n logn)

TAMAT