ANALISIS KOMPLEKSITAS ALGORITMA PENCARIAN *CONVEX HULL* PADA BIDANG PLANAR

Petra Novandi – NIM : 13505059

Program Studi Teknik Informatika, Institut Teknologi Bandung
Jl. Ganesha 10, Bandung
E-mail : if15059@students.if.itb.ac.id
Website : http://students.if.itb.ac.id/~if15059/, http://van-odin.net/

ABSTRAK

Makalah ini membahas tentang analisis kompleksitas algoritma pencarian *convex hull* dari himpunan titik terbatas pada bidang planar. *Convex hull* sebuah himpunan titik pada bidang planar adalah himpunan titik – titik yang membentuk sebuah poligon konveks yang melingkupi seluruh himpunan titik tersebut. Algoritma yang telah ditemukan untuk memecahkan permasalahan ini antara lain *Brute Force*, *Jarvis' March*, *Quick Hull*, *Divide and conquer, Monotone Chain*, *Incremental*, *Mariage before Conquest*, dan lain – lain.

Adapun dari banyak algoritma yang telah ditemukan yang akan dianalisa di sini adalah *Brute Force*, *Jarvis March*, *Graham Scan*, dan *Divide and Conquer*. Analisa yang dilakukan pada algoritma-algoritma ini adalah untuk mencari kompleksitas waktu asimtotiknya. Pencarian kompleksitas tersebut dilakukan dengan menghitung kompleksitas *upper bound* atau yang sering disebut Big O dari masing-masing algoritma menggunakan teorema *Big O* untuk algoritma sekuensial dan *Master Theorem* untuk algoritma rekurens. Analisa ini juga mencari kasus terbaik dan kasus terburuk dari setiap algoritma.

Sejak pencarian *convex hull* banyak diperlukan dalam kehidupan sehari-hari dan sejalan dengan perkembangan teknologi yang memerlukannya untuk hal – hal yang menyangkut kehidupan orang banyak, maka sangat pentinglah untuk mencari pemakaian algoritma yang sesuai, efektif, dan efisien. Analisa ini dilakukan untuk mencari daya guna dan kemangkusan dari algoritma-algoritma tersebut meski tidak bermaksud untuk membandingkan daya guna dan kemangkusan masing-masing untuk menyelesaikan masalah.

Kata Kunci : *Algorithm Analysis, Computational Geometry, Convex hull,* Kompleksitas algoritma

1. Pendahuluan

Banyak sekali hal di dalam kehidupan sehari – hari yang sangat berhubungan dengan geometri. Segala bentuk dan wujud yang ada di sekitar manusia didefinisikan ke dalam geometri. Misalnya untuk menentukan bentuk sebuah benda, untuk menentukan dua buah terdekat dari benda-benda yang ada. Sejalan dengan perkembangan teknologi, masalah – masalah menjadi sangat kompleks dan menyangkut kehidupan orang banyak seperti sistem navigasi *autopilot* pesawat terbang dalam menentukan bentuk permukaan bumi yang akan dilaluinya. Masalah – masalah tersebut sangat memerlukan algoritma yang dirancang dengan baik untuk menyelesaikannya dengan efektif dan efisien. Pada sekitar tahun 1970 disiplin ilmu geometri komputasional *(computational geometry)* untuk membantu menyelesaikan masalah-masalah tersebut.

Computational Geometry adalah salah satu bidang ilmu komputer yang mempelajari tentang masalah – masalah geometri. Di dalam dunia rekayasa modern dan matematika, aplikasi – aplikasi tentang geometri komputasional telah banyak digunakan pada disiplin – disiplin lain seperti dalam *computer graphic*, robotik, desain VLSI, CAD (*Computer Aided Design*), dan juga statistik.

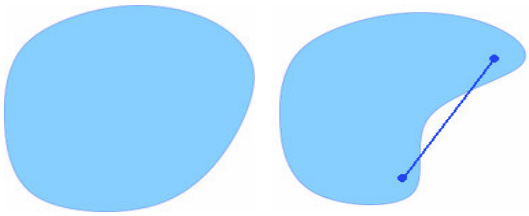
Permasalahan geometri komputasional biasanya melingkupi objek – objek geometri seperti himpunan titik, himpunan garis, himpunan segmen, dan juga poligon. Beberapa bahkan ada permasalahan geometri yang muncul dari disiplin ilmu tersebut.

Salah satu cabang dari *computational geometry* adalah *combinatorial computational geometry* yang juga dikenal dengan *algorithmic geometry*. Tujuan utama cabang ilmu ini adalah untuk mengembangkan algoritma yang efisien dan struktur data yang tepat untuk menyelesaikan masalah-masalah yang berhubungan dengan geometri. Disiplin ini menjadi penting untuk disiplin ilmu lain seperti GIS dan *computer graphic* di mana sekarang ini komputer dipakai untuk memproses puluhan dan ratusan milyar data yang ada.

Salah satu permasalahan manusia yang berkaitan dengan penentuan bentuk geometri dapat diselesaikan dengan menggunakan algoritma pencarian *convex hull*. Salah satu aplikasi penggunaan *convex hull* adalah untuk menentukan bentuk permukaan alam seperti yang sering dipakai dalam disiplin ilmu GIS *(Geographical Information System)*

2. *Convex hull*

Sebuah subset S dari bidang $\mathbb{R}$ disebut konveks jika dan hanya jika pada seluruh dua buah titik sembarang $p, q \in S$ dibentuk garis yang seluruhnya berada dalam S[1].

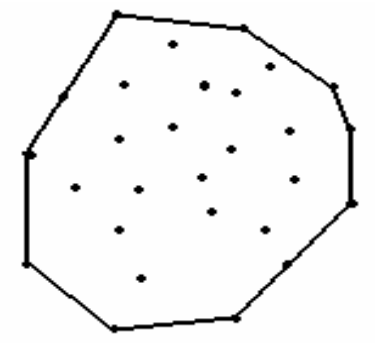


Gambar 1 Convex Set

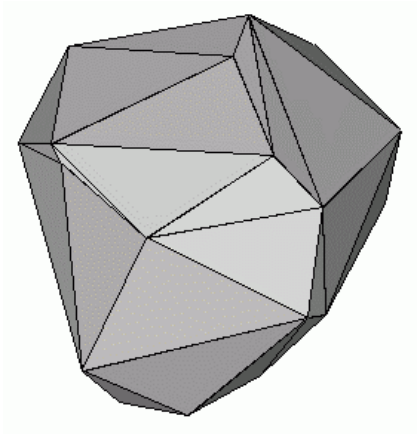
Pencarian *convex hull* dari sebuah himpunan titik Q (CH(Q)) adalah mencari sebuah convex set terkecil yang memuat seluruh titik pada Q.

Convex hull dari dari sebuah himpunan titik Q (CH(Q)) pada n dimensi adalah seluruh irisan dari semua convex set yang mengandung Q. Terlebih lanjut, untuk N buah titik $p_1, p_2, \ldots p_N$. *convex hull* merupakan himpunan *convex combination* yang dinyatakan dengan

$$CH(Q) \equiv \left\{ \sum_{j=1}^{N} \lambda_j p_j ; \lambda_j \geq 0; \sum_{j=1}^{N} \lambda_j = 1 \right\}$$



Gambar 2 *Convex hull* 2D



Gambar 3 *Convex hull* 3D

3. Algoritma Pencarian *Convex hull*

Usaha pencarian algoritma yang efisien untuk pencarian *convex hull* sudah dilakukan sejak lama dan masih dilakukan sampai sekarang. Berikut adalah daftar algoritma pencarian *convex hull* pada bidang planar yang telah ditemukan beserta penemu dan tahun publikasi pertamanya[2].

Daftar Algoritma *Convex hull* Berdasarkan Tahun Publikasi

No	Algoritma	Penemu	Tahun
1.	Brute Force	N/A	-
2.	Graham Scan	Graham	1972
3.	Jarvis March	Jarvis	1973
4.	Divide-and-Conquer	Preparata & Hong	1977

5.	Monotone Chain	Andrew	1979
6.	Incremental	Kallay	1984
7.	Marriage-before-Conquest	Kirkpatrick & Seidel	1986

Sumber : http://www.geometryalgorithms.com/Archive/algorithm_0109/algorithm_0109.htm

Dari seluruh algoritma yang telah didaftar di atas, yang akan dianalisis di sini antara lain *Brute Force*, *Graham Scan*, *Jarvis March*, dan *Divide and Conquer*.

3.1. *Brute Force*

Dalam ilmu komputer, *brute force search* atau *exhaustive search* adalah cara trivial dan teknik umum untuk memecahkan masalah. Brute force search secara sederhana mengenumerasi seluruh kemungkinan kandidat yang bisa menjadi solusi masalah dan mememeriksa apakah kandidat-kandidat tersebut memenuhi solusi permasalahan.

Brute force search adalah cara yang sangat mudah diimplementasikan untuk memecahkan masalah karena brute force search selalu mencari keberadaan solusi dari seluruh kemungkinan yang ada. Akan tetapi pemecahan masalah menggunakan metode brute force biasanya memerlukan sumber yang banyak baik waktu dan memori dan kebutuhan tersebut makin membesar dengan cepat sejalan dengan banyaknya permasalahan yang ingin diselesaikan.

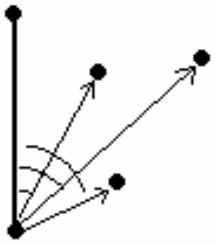
Biasanya cara ini digunakan jika banyaknya permasalahan dibatasi untuk jumlah yang tidak terlalu banyak.

3.1.1. Algoritma

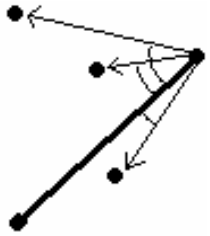
Pencarian *convex hull* dengan menggunakan Brute Force ada 2 macam: *Naive Brute Force* dan *Better Brute Force*. Naive Brute Force mencari *convex hull* dengan cara mengiterasi seluruh titik yang ada dan mememeriksa apakah titik tersebut berada di dalam seluruh kombinasi segitiga yang dapat dibentuk oleh tiga buah titik dari titik yang lain. Sebuah titik merupakan anggota *convex hull* jika titik tersebut tidak terletak di dalam semua kombinasi segitiga lain tersebut. Berbeda dengan Naive Brute Force, Better Brute Force mencari *convex hull* dengan mencari semua kombinasi garis yang dapat dibentuk dari dua buah titik pada himpunan Q dan kemudian memeriksa apakah garis tersebut merupakan *convex hull* yakni dengan membandingkan keberadaan titik lain menggunakan *cross product*.

Langkah – langkah pencarian dengan menggunakan Better Brute Force adalah

1. Untuk setiap titik p_1 pada himpunan Q, bentuk vektor $p_1 p_2$ dengan semua titik lain pada Q.



(a) *Convex hull*



(b) Bukan *Convex hull*

Gambar 4 Brute Force

2. Pada vektor $p_1 p_2$, bandingkan keberadaan vektor tersebut dengan titik p_3 yakni seluruh titik pada Q selain $p_1 p_2$. Untuk membandingkan tiga titik tersebut, diperlukan cross product dari dua buah vektor yang dibentuk dari tiga titik tersebut.
3. Jika seluruh titik p_3 berada di sebelah kiri atau sebelah kanan vektor p_1p_2, maka titik p_1 dan p_2 merupakan CH(Q).

```
1   BruteForce(Q,n)
2     for i ← 1 to n do
3       for j ← 1 to n do
4         if (j ≠ i) then
5           palingKiri ← true
6           palingKanan ← true
7           for j ← 1 to n do
8             if (j ≠ i) and (j ≠ i)
      then
9               if (isLeft(Q[i], Q[j],
      Q[k])) then
10                palingKiri ← false
11              else
12                palingKiri ← false
13          if      (palingKiri)      or
              (palingKanan) then
12            Tambah(CH, Q[i], Q[j])
          {kedua titik convex hull}
13    return CH {CH adalah array yang
        berisi Convex hull}
```

Pseudocode Brute Force

3.1.2. Analisis

Metode Brute Force ini menggunakan tiga buah iterasi utama. Iterasi terluar pada baris 2 digunakan untuk mencacah titik lalu iterasi selanjutnya digunakan untuk mencacah titik – titik lain untuk membentuk dua garis. Lalu iterasi di dalamnya digunakan untuk membandingkan letak titik-titik selain kedua titik itu dengan garis yang dibentuk.

Seperti yang ditulis di atas, pembandingan tersebut menggunakan *cross product*. Kompleksitas pencarian ini menggunakan **O(1).** Pencarian tersebut hanya menggunakan operasi – operasi matematika untuk menentukannya.

```
1   isLeft(p1, p2, p3)
2     return (dotProduct(p1, p2, p3) >
        0)

1   CrossProduct(p1, p2, p3) : real
2     return ((p2.x - p1.x)*(p3.y -
          p1.y) - (p3.x - p1.x)*(p2.y
          - p1.y))
```

Pada iterasi terdalam dari keseluruhan, ekspresi yang dieksekusi bernilai

$$T(n) = O(1) + O(1) + O(1)$$

$$T(n) = \max(O(1), O(1), O(1))$$

$$T(n) = O(1)$$

masing-masing O(1) adalah untuk pengecekan, pengubahan nilai boolean dan penambahan titik ke dalam himpunan CH(Q) (jika ada). Oleh karena itu nilai keseluruhan dari algoritma adalah

```
for i ← 1 to n do          O(n)
  for j ← 1 to n do        O(n)
    if (j ≠ i) then
      palingKiri ← true    O(1)
      palingKanan ← true   O(1)
      for j ← 1 to n do    O(n)
        {}                 O(1)
```

$$T(n) = O(n)O(n)(O(1) + O(1) + (O(n)O(1)))$$

$$T(n) = O(n.n)\max(O(1), O(1), O(n))$$

$$T(n) = O(n.n.n)$$

$$T(n) = O(n^3)$$

Kompleksitas waktu asimtotik yang digunakan $\mathbf{O(n^3)}$ Karena setiap operasi dilakukan dengan waktu yang konstan maka dapat disimpulkan juga bahwa T(n) = $\mathbf{\Theta(n^3)}$

3.2. *Graham Scan*

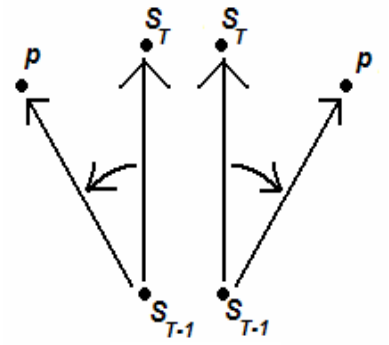
Pada tahun 1972 Ronald Lewis Graham, seorang matematikawan berkebangsaan Amerika Serikat, mempublikasikan sebuah algoritma efisien untuk kasus planar[3]. Dalam sejarah ini merupakan publikasi pertama yang menunjukkan bahwa pencarian *convex hull* dalam kompleksitas waktu asimtotik O(n log n). Pada tahun 1981 A.C. Yao membuktikan bahwa algoritma tersebut adalah yang paling optimal dalam kasus terburuk. Karena kecepatannya tersebut algoritma ini adalah yang paling populer dalam pencarian *convex hull* dalam bidang planar. Sayangnya algoritma ini tidak bisa digunakan dalam ruang berdimensi di atas planar (>2). Ini dikarenakan Graham Scan menggunakan pengurutan titik berdasarkan sudut di mana belum ada metode yang analog pada dimensi di atas dua.

3.2.1. Algoritma

Langkah – langkah *Graham Scan* untuk menentukan *convex hull* :

1. Tentukan pivot $\boldsymbol{p_0}$ satu titik yang terdapat pada koordinat *y* terkecil dalam himpunan Q. Jika ada lebih dari satu titik yang memiliki ordinat (koordinat *y*) yang sama maka $\boldsymbol{p_0}$ adalah titik yang memiliki absis (koordinat *x*) terkecil.
2. Urutkan seluruh titik tersisa pada himpunan Q menjadi $\boldsymbol{p_1}$ p_2 ... $\boldsymbol{p_{n-1}}$ dari titik yang berada relatif paling kanan pada sistem koordinat polar terhadap $\boldsymbol{p_0}$

3. Masukkan p_0 dan p_1 ke dalam tumpukan S sebuah himpunan titik pada Q yang menjadi *convex hull*.
4. Untuk p setiap titik tersisa dari p2 sampai pn, bandingkan vektor antara p dan titik kedua *teratas* pada tumpukan S (S_{T-1}) dengan titik teratas teratas (S_T) dan S_{T-1}. Jika vektor pertama terletak relatif lebih kanan maka S_T pada tumpukan S bukan termasuk **CH(Q)** dan kemudian dikeluarkan dari tumpukan S. Masukkan p ke dalam S.



Gambar 5

5. Tumpukan S merupakan **CH(Q)**

Pseudocode untuk algoritma ini adalah

```
1  GrahamScan(Q, n)
2    CariPivot(Q)
3    SortTerhadapPivot(Q)      {Q(0)
       adalah pivot}
4    Push(S, Q[0])
5    Push(S, Q[1])
6    for i 2 to n - 1 do
7      if   (IsKanan(Q[i],   Top(S),
           Second(S)) then
8        Pop(S)
9      Push(S, Q(i))
10   return S        {S        berisi
       seluruh titik convex hull}
```

Pseudocode Graham Scan

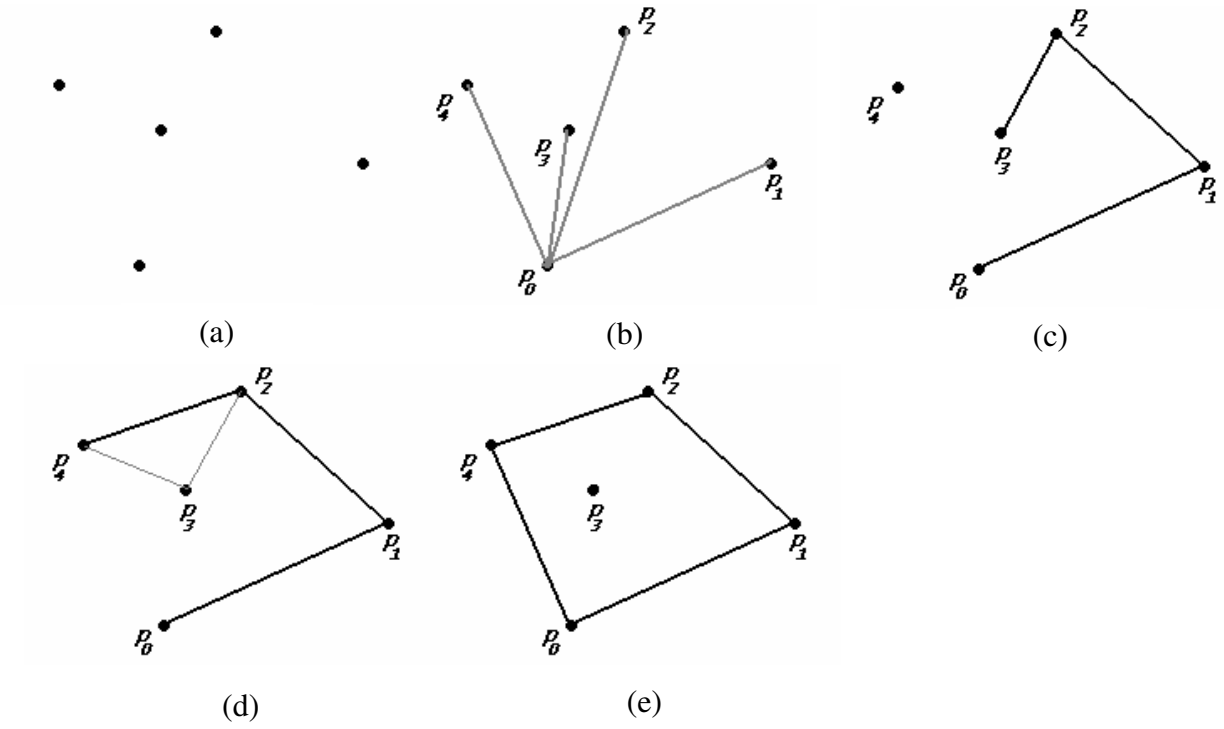
3.2.2. Analisis

Sekarang akan dicari kompleksitas asimptotik dari fungsi Graham Scan.

Algoritma ini hanya memiliki sebuah kalang traversal. Tepat di dalam kalang tersebut, masing-masing eksekusi yang dilakukan tepat memiliki kompleksitas **O(1)**. Untuk mengetahui apakah sebuah sebuah titik adalah membelok kanan dari dua titik lain hanya memerlukan waktu **O(1)**. Seperti pada perhitungan algoritma *Brute Force Search*, perhitungan tersebut tidak memerlukan pencarian beda sudut pada segmen antara dua vektor yang dibentuk tapi hanya dengan perhitungan biasa yakni mencari cross product dari ketiga titik.

Untuk tiga titik (x_1, y_1), (x_2, y_2), dan (x_3, y_3), diberikan hasil cross product nya adalah

$$C = ((x_2 - x_1)(y_3 - y_1) - (x_3 - x_1)(y_2 - y_1))$$



Gambar 6. Graham Scan

Jika hasil ini bernilai negatif disimpulkan bahwa titik vektor yang dibentuk kemudian berada lebih kanan dari vektor antara dua titik yang berada paling atas pada S. Dan jika hasilnya bernilai 0 maka menandakan bahwa ketiga titik berada dalam satu garis.

```
1  isKanan(p1, p2, p3)
2    return (dotProduct(p1, p2, p3) <
       0)

1  CrossProduct(p1, p2, p3) : real
2    return ((p2.x - p1.x)*(p3.y -
         p1.y) - (p3.x - p1.x)*(p2.y
         - p1.y))
```

Iterasi pada baris 6 dieksekusi selalu dieksekusi sebanyak n – 3 kali. Karena fungsi Push() dan Pop() untuk stack masing-masing menggunakan waktu **O(1)** kali, pengecekan vektor juga memerlukan O(1) maka waktu yang digunakan untuk mengeksekusi loop

$$T(n) = n(O(1) + O(1) + O(1))$$

$$T(n) = nO(1)$$

$$T(n) = O(n)$$

adalah O(n).

Pencarian Pivot pada himpunan titik Q yang dieksekusi pada baris 2 adalah dengan mengiterasi setiap titik yang ada dan mencari titik yang berada pada ordinat terkecil. Operasi ini memerlukan O(n) kali untuk perbandingan dimana n adalah banyak titik yang ada pada Q.

Setelah pencarian pivot tersebut, seluruh titik yang tersisa di urutkan berdasarkan letaknya relatif terhadap pivot. Pada baris 3 langkah tersebut dieksekusi. Pengurutan dapat dilakukan dengan menggunakan *mergesort* atau pun *heapsort* dan untuk membandingkan titik dapat mencari crossproduct dari dua titik yang dibandingkan dengan titik yang menjadi pivot. Pengurutan ini menggunakan waktu rata-rata **O(n log n)**

```
CariPivot(Q)                    O(n)
SortTerhadapPivot(Q)          O(nlogn)
Push(S, Q[0])                   O(1)
Push(S, Q[1])                   O(1)
for i 2 to n – 1 do             O(n)
    {}
```

Jadi total waktu yang digunakan pada algoritma Graham Scan adalah **O(n log n)**. Pada kasus terburuk yakni jika seluruh titik adalah *convex hull* algoritma ini Akan tetapi pada kasus terbaik yakni jika masukan himpunan titik Q sudah terurut lebih dahulu maka proses pengurutan dapat diabaikan dan kompleksitasnya menjadi tepat **Θ(n)** karena titik – titik ini tepat diiterasi seluruhnya untuk mencari titik yang merupakan *convex hull*.

3.3. *Jarvis March*

R.A. Jarvis pada 1973 mempublikasikan algoritma yang kemudian dinamakan sesuai dengan namanya[4]. Algoritma ini sering dikenal dengan Gift Wrapping karena sesuai namanya, algoritma ini analog dengan cara sehari – hari manusia dalam membungkus kado. Secara intuitif Jarvis March mensimulasikan pembungkusan kertas himpunan titik Q. Dimulai dari titik yang mempunyai ordinat terendah lalu kertas ditarik ke kanan sampai menyentuh sebuah titik yang kemudian dianggap sebagai *convex hull*, kemudian diulangi lagi sampai menemukan titik yang pertama.

3.3.1. Algoritma

Langkah-langkah untuk mencari *convex hull* pada algoritma ini

1. Tentukan titik ekstrim yang memiliki absis terkeci yang menjadi titik pertama pada *convex hull*
2. Cari titik selanjutnya yang di mana garis antara titik tersebut dengan titik sebelumnya berada di sebelah kiri semua titik yang ada
3. Titik yang didapat tersebut adalah anggota *convex hull* kemudian ulangi langkah di atas untuk mencari *convex hull* selanjutnya

Pseudocode untuk algoritma ini

```
1  JarvisMarch(Q)
2  i ← 1
3  CH[i] ← TitikTerendah(Q)
4  repeat
5    CH[i+1] ←
       CariConvexHull(Q,CH[i])
6    i ← i + 1
7  until (CH[i] = CH[i-1])
```

Pseudocode Jarvis' March

3.3.2. Analisis

Pada iterasi pertama di baris 5 algoritma tersebut mencari semua titik yang menjadi *convex hull*.

Cara pencarian *convex hull* selanjutnya ada dua macam :

1. Mencoba setiap titik yang ada dan memeriksa apakah garis yang dibentuk titik tersebut dengan titik *convex hull* sebelumnya merupakan CH(Q) yakni dengan membandingkannya dengan seluruh titik yang ada. Cara ini sama seperti dengan metode brute force search yang

telah ditunjukkan sebelumnya. Langkah ini membutuhkan **O(n^2)** dalam pengeksekusiannya.
Misalkan *h* adalah banyaknya CH(Q), maka total waktu yang dibutuhkan

$$T(n) = hO(n^2)$$

$$T(n) = O(h)O(n^2)$$

$$T(n) = O(hn^2)$$

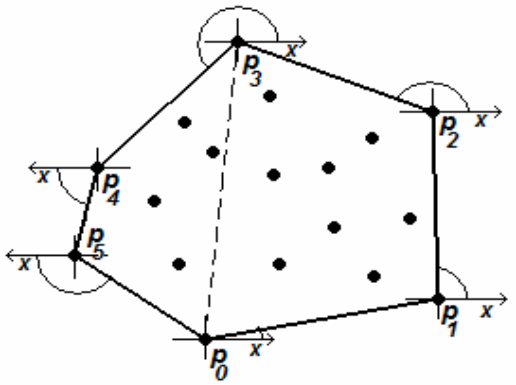
Kasus terburuk untuk algoritma ini adalah jika $h = n$ maka $T(n) = O(n^3)$. Sedangkan kasus terbaiknya adalah ketika $h = \log n$ yakni $T(n) = O(n^2 \log n)$

2. Cara yang ini membangun runtunan CH(Q) = p_o, p_1, p_2, ... p_h dengan membandingkan sudut polar. Misalkan titik pertama adalah p_o, maka p_1 adalah titik dengan sudut polar terkecil terhadap p_o. p_2 merupakan titik dengan sudut polar terkecil terhadap p_2 dan seterusnya. Jika ada titik yang sama sudutnya pilih yang terjauh. Setelah runtunan mencapai titik tertinggi, pk maka algoritma telah membentuk bagian kanan dari *convex hull* tersebut. Untuk mendapatkan p_{k+1} maka cari titik yang memiliki sudut polar terkecil dari p_k tetapi dari sumbu x negatif. Untuk mencari sudut terkecil hanya diperlukan total waktu ***O(n)*** karena waktu untuk mencari sudut polar tersebut adalah ***O(1)***. Maka keseluruhannya adalah

$$T(n) = hO(n)$$

$$T(n) = O(h)O(n)$$

$$T(n) = O(hn)$$



Gambar 7 Jarvis' March

Sama seperti cara pada poin 1, algoritma ini memiliki kasus terbaik jika $h = n$ maka $T(n) = O(n^2)$ Sedangkan kasus terbaiknya adalah ketika $h = \log n$ yakni $T(n) = O(n \log n)$

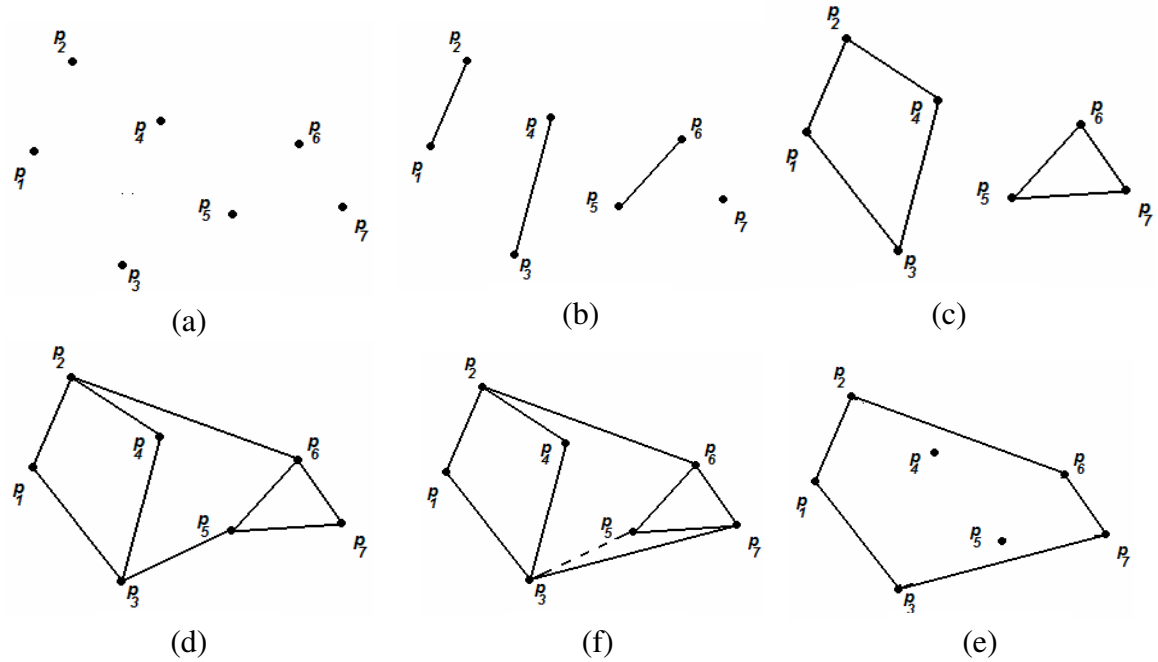
3.4. *Divide and Conquer*

Divide and conquer adalah sebuah desain algoritma yang sangat penting di dalam ilmu komputer. Dalam sebuah permasalahan, algoritma ini bekerja secara rekursif untuk membagi permasalahan tersebut menjadi sub – sub permasalahan sampai sub permasalahan ini dapat diselesaikan secara langsung. Solusi dari permasalahan kecil ini kemudian dikombinasikan dengan solusi sub permasalahan lain untuk menyelesaikan permasalahan yang lebih besar sampai keseluruhan permasalahan diselesaikan.
Teknik ini menjadi basis dari banyak algoritma efisien untuk berbagai permasalahan seperti pengurutan dan juga pencarian *convex hull*.

3.4.1. Algoritma

Cara utama yang dilakukan oleh algoritma ini untuk membentuk sebuah *convex hull* dari himpunan titik adalah dengan membagi dua semua himpunan menjadi 2 subhimpunan dan mencari *convex hull* dengan cara yang sama sampai subhimpunan terdiri dari 2 atau 1 buah titik lalu semua 2 buah sub himpunan tersebut digabungkan sampai keseluruhan Q terdapat dalam sebuah *convex set*.

1. Urutkan setiap titik dalam Q berdasarkan absisnya.
2. Bagi semua titik sama banyak ke dalam 2 sub himpunan Q dan secara rekursif cari *convex hull* dari masing – masing sub himpunan sampai sub himpunan terdiri dari 1 atau 2 buah titik.
3. Gabungkan dua buah himpunan *convex hull* yang menjadi sub himpunan Q,
   a. Hubungkan titik tertinggi dari kedua himpunan *convex hull*
   b. Iterasi setiap titik searah jarum jam untuk himpunan yang berada di sebelah kanan, dan berlawanan arah jarum jam pada himpunan yang berada di sebelah kiri.
   c. Hubungkan titik terendah dari kedua himpunan dan gunakan cara yang sama.



Gambar 8 Divide And Conquer

```
1   DivideAndConquer(Q)
2     Sort(Q)
3     return ConvexHull(Q)
4   ConvexHull(Q)
5     if (NB(Q) = 1) or (NB(Q) = 2))
        then {basis}
6       return Q
7     else
8       Divide(Q, Q1, Q2)
9       Q1 ← ConvexHull(Q1)
10      Q2 ← ConvexHull(Q1)
11      return Merge(Q1, Q2)
12
13  Divide(Q, Q1, Q2)
14    n ← NB(Q)
15    for i ← 1 to Floor(n/2) do
16      Q1[i] ← Q[i]
17    for i ← Floor(n/2) + 1 to n do
18      Q2[i] ← Q[i - Floor(n/2) + 2]
```

Pseudocode Divide And Conquer

3.4.2. Analisis

Berikut akan dianalisis kompleksitas waktu untuk mencari CH(Q) dengan menggunakan algoritma *divide and conquer*

Pada baris 2, seluruh titik Q diurutkan berdasarkan absisnya. Proses pengurutan ini dapat menggunakan pengurutan quicksort yang mempunyai kompleksitas waktu ***O(n log n)***

Untuk membagi dua sebuah himpunan dibutuhkan O(n) karena hanya mengiterasi seluruh titik yang sudah terurut pada Q. Demikian juga menggabungkan 2 buah himpunan membutuhkan waktu asimtotik linier karena dalam *convex hull* sudah otomatis terurut berdasarkan letaknya.

Fungsi waktu untuk fungsi `ConvexHull` adalah

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

Dimana O(n) adalah kompleksitas waktu untuk membagi himpunan menjadi dua sub himpunan dan mengabungkan dua buah *convex hull* yang telah dicari.

Untuk mencari kompleksitas waktu asimptotik dari bentuk rekurens seperti ini dapat digunakan cara *Master Theorem*[5]

Pada bentuk $T(n)=aT(n/b)+f(n)$, jika

1) $f(n)\in O(n^{\log_b a-\varepsilon}),\varepsilon>0$ maka $T(n)=\Theta(n^{\log_b a})$
2) $f(n)\in \Theta(n^{\log_b a})$ maka $T(n)=\Theta(n^{\log_b a}\log n)$
3) $f(n)\in O(n^{\log_b a+\varepsilon}),\varepsilon>0$ dan $af(\frac{n}{b})\le cf(n),c>1$ maka $T(n)=\Theta(n^{\log_b a})$

Dari fungsi waktu ConvexHull didapat

$a=2,b=2$

Karena

$$f(n)=O(n)\in \Theta(n^{\log_2 2})=\Theta(n)$$

maka fungsi waktu dari bentuk rekurens fungsi ConvexHull adalah

$$T(n) = \Theta(n \log n)$$

Waktu pencarian *convex hull* sama dengan waktu untuk mengurutkan titik yang ada. Dengan demikian kompleksitas waktu asimptotik dari metode divide and conquer adalah ***O(n log n)***

4. Kesimpulan

Dari hasil analisa kompleksitas waktu dapat disimpulkan untuk n adalah banyaknya titik yang ada pada tiap kasus

1. Algoritma pencarian CH(Q) dengan menggunakan Brute Force Search adalah algoritma yang paling stabil karena untuk seluruh kasus yang adanya, semuanya dilakukan dengan waktu yang sama. Brute Force search juga sangat mudah diimplementasikan. Akan tetapi algoritma ini menggunakan waktu yang sangat lama yakni

   $$T(n) = \Theta(n^3)$$

2. Algoritma Jarvis' March mempunyai kompleksitas waktu

   $$T(n) = O(nh)$$

   di mana h adalah banyaknya CH(Q) yang ada pada setiap kasus.
   Adapun kasus terbaik (*best case*) adalah jika *h* = *log n* yang memberikan

   $$T(n) = O(n \log n)$$

   Dan kasus terburuk (*worst scenario*) pada saat h = n (seluruh titik merupakan CH(Q)) yang memberikan

   $$T(n) = O(nh)$$

3. Algoritma Graham Scan memiliki kompleksitas

   $$T(n) = O(n \log n)$$

   Nilai ini didapat dari pengurutan titik yang dilakukan terlebih dahulu. Dapat dikatakan algoritma ini bergantung pada kompleksitas pengurutan yang digunakan. Untuk kasus terbaik yakni himpunan Q sudah diurutkan terlebih dahulu, maka pengurutan dapat dilewatkan dan memberikan

   $$T(n) = O(n)$$

   Sayangnya algoritma ini hanya bisa digunakan pada bidang berdimensi 2.

4. Divide and conquer bertujuan untuk mencari CH(Q) secara rekursif. Kompleksitas waktu rata-ratanya adalah

   $$T(n) = \Theta(n \log n)$$

   Algoritma ini tidak memiliki contoh kasus terbaik dan terburuk

DAFTAR PUSTAKA

[1] MathWorld, Wolfram Research. http://mathworld.wolfram.com/ConvexSet.html; Tanggal akses pada 2 Januari 2007.

[2] Geometry Algorithm. http://www.geometryalgorithms.com/Archive/algorithm_0109/algorithm_0109.htm; Tanggal akses pada 2 Januari 2007

[3] R. L. Graham (1972). *An efficient algorithm for determining the convex hull of a finite planar set* Information Processing Letters

[4] R. A. Jarvis, "On the identification of the *convex hull* of a finite set of points in the plane" Inform. Process. Lett., 2, 1973, 18--21.

[5] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Second Edition*. MIT Press and McGraw-Hill, 2001. ISBN 0-262-03293-7