# Optimasi Genomic Sequence Alignment dengan Pendekatan Divide and Conquer menggunakan Algoritma Hirschberg

Ignatius Jhon Hezkiel Chan- 13522029[1]
*Program Studi Teknik Informatika*
*Sekolah Teknik Elektro dan Informatika*
*Institut Teknologi Bandung, Jl. Ganesha 10 Bandung 40132, Indonesia*
*[1]13522029@std.stei.itb.ac.id*

***Abstraksi*—Genomic Sequence Alignment adalah metode dalam bidang bioinformatika untuk menyelaraskan rangkaian DNA, RNA, atau protein dengan tujuan mengidentifikasi area kesamaan yang menandakan hubungan antar organisme. Penyelarasan ini esensial dalam penelitian biologi molekuler, evolusi, dan medis. Masalah alignment umumnya dapat diselesaikan hanya dengan pendekatan dynamic programming, seperti dengan algoritma Needleman-Wunsch Algorithm. Optimasi dapat dilakukan pada implementasi tersebut dengan menerapkan konsep Divide and Conquer menggunakan Algoritma Hirschberg. Dengan pendekatan ini, penyelarasan global dapat dilakukan dengan tingkat ketelitian yang tinggi dan dengan kompleksitas ruang yang lebih efisien dibandingkan hanya dengan DP sehingga memungkinkan analisis yang lebih akurat dan mendalam terhadap urutan genom khususnya sekuens-sekuens genom yang sangat panjang.***

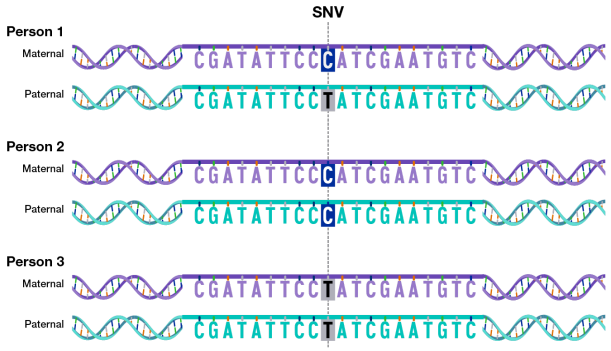
***Keywords*—genomic sequence, Hirschberg, Needleman Wunsch**

## I. Pendahuluan

Mengidentifikasi kesamaan atau perbedaan antar organisme merupakan hal yang penting dalam ilmu biologi dan medis. Kesamaan atau perbedaan ini dapat dilihat dari urutan DNA, RNA, atau protein antara berbagai organisme. Dengan memahami bagaimana gen tertentu dipertahankan atau bervariasi di antara spesies yang berbeda, kita akan mendapat wawasan baru tentang fungsi dan evolusi dari gen tersebut serta hubungan antarspesies.

Genomic sequence (urutan genomik) merupakan representasi genom dalam bentuk rangkaian huruf yang menyimpan semua informasi genetik dari suatu organisme. Rangkaian huruf ini memiliki tiap karakter yang melambangkan suatu komponen dari urutan genomik, yakni basa nitrogen (A,T, C, G) atau nukleotida (A, U, C, G) dalam DNA atau RNA suatu organisme. Tiap urutan basa nitrogen dan nukleotida dalam urutan genomik ini mencerminkan susunan gen yang membentuk karakteristik genetik organisme tersebut.

Genomic Sequence Alignment merupakan aspek fundamental dalam bioinformatika, yang berfungsi untuk membandingkan dan menganalisis urutan DNA atau RNA dari berbagai organisme. Penyelarasan ini berguna agar dapat lebih mudah menganalisis urutan gen serta mengidentifikasi kesamaan dari variansi sequence. Penyelarasan dari urutan gen ini menjadi langkah dasar yang diperlukan untuk memastikan bahwa analisis lebih lanjut dapat dilakukan dengan tepat dan akurat. Tanpa penyelarasan yang benar, hasil analisis genomik dapat menjadi tidak akurat atau bahkan menyesatkan, sehingga menghambat kemajuan penelitian dan aplikasi klinis.




*Gambar 1 Contoh Genomic Sequence pada beberapa manusia(sumber: https://www.genome.gov/about-genomics/educational-resources/fact-sheets/human-genomic-variation)*

Pemrograman Dinamis (DP) telah memainkan peran revolusioner dalam penelitian genomik dan pengobatan personal dengan menyediakan metode yang efisien dan akurat untuk menyelaraskan urutan genomik. Algoritma DP, seperti Needleman-Wunsch memungkinkan para ilmuwan untuk melakukan penyelarasan global dan lokal dengan tingkat ketelitian yang tinggi.

Akan tetapi, pada kenyataannya panjang sekuens-sekuens genomik sangat besar karena menyangkut suatu organisme ataupun kumpulan dari organisme. Hal ini menyebabkan aplikasi DP pada sekuens yang sangat panjang membutuhkan waktu dan resource komputasi yang sangat besar. Hal ini tentu dapat menghambat jalannya penelitian karena jumlah resource yang terpakai sangat besar yang secara tidak langsung mempengaruhi performa mesin.

Aplikasi DP sebelumnya dapat dilakukan optimasi lebih lanjut dengan menerapkan konsep *Divide and Conquer* melalui algoritma Hirschberg. Dengan algoritma ini, efisiensi

ruang akan meningkat sehingga pemrosesan untuk sekuens-sekuens yang panjang dapat dilakukan dengan cepat dan dengan ruang yang lebih efisien. Hal ini memungkinkan analisis genomik yang lebih scalable, terutama dalam konteks proyek-proyek genomik besar seperti analisis genom populasi.

## II. Landasan Teori

### A. *Dynamic Programming*

Dynamic programming (DP) adalah teknik algoritma yang digunakan untuk memecahkan masalah optimasi, dimana suatu masalah diurai menjadi sekumpulan tahap sedemikian sehingga solusi persoalan dapat dipandang sebagai serangkaian keputusan yang saling berkaitan.

Rangkaian keputusan optimal pada pendekatan ini menggunakan prinsip optimalitas, yakni jika solusi total optimal maka bagian solusi sampai tahap ke-k juga optimal. Pendekatan ini perlahan menyusun solusi optimal dari submasalah bertahap hingga membentuk solusi optimal utama sesuai dengan prinsip optimalitas.

### B. *Divide and Conquer*

Divide and Conquer (DNC) merupakan salah satu paradigma algoritma dengan membagi masalah yang kompleks menjadi submasalah yang lebih kecil, menyelesaikan masing-masing sub masalah secara terpisah, dan kemudian menggabungkan solusi solusi tersebut untuk mendapatkan solusi dari submasalah yang asli.

Pendekatan ini dibagi menjadi tiga langkah utama, yakni

1. *Divide*

   Pada langkah ini, masalah besar dibagi menjadi dua atau lebih submasalah yang lebih kecil. Pembagian ini harus dilakukan sehingga setiap submasalah memiliki struktur yang sama dengan masalah aslinya, tetapi dengan ukuran yang lebih kecil.

2. *Conquer*

   Setelah masalah dibagi menjadi submasalah yang lebih kecil, langkah selanjutnya adalah menyelesaikan masing-masing sub masalah secara terpisah. Ini dilakukan dengan cara menerapkan algoritma atau metode yang sesuai untuk menyelesaikan sub masalah tersebut.

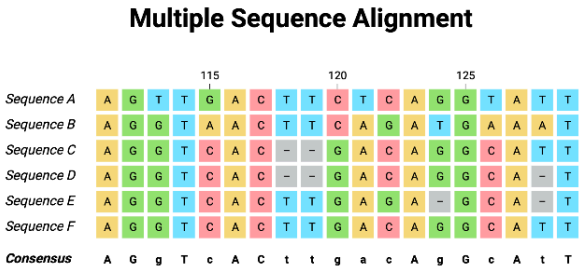
3. *Combine*

   Setelah semua sub masalah diselesaikan, langkah terakhir adalah menggabungkan solusi-solusi tersebut untuk mendapatkan solusi dari masalah aslinya.

### C. *Alignment Sekuens*

Alignment sekuens adalah proses membandingkan dua atau lebih sekuens biologis, seperti sekuens DNA, RNA, atau protein, untuk menemukan kesamaan atau perbedaan di antara mereka. Terdapat dua jenis utama dari alignment sekuens , yakni alignment global yang membandingkan keseluruhan sekuens, dan alignment lokal yang hanya membandingkan bagian tertentu dari kedua sekuens.

Proses alignment sekuens biasanya melibatkan penggunaan matriks skor atau matriks substitusi, di mana setiap sel dalam matriks memberikan skor untuk pasangan karakter yang sesuai atau tidak sesuai. Selanjutnya, algoritma alignment digunakan untuk menghitung jalur optimal atau solusi alignment berdasarkan skor-skor dari matriks tersebut.




*Gambar 2 Contoh Multiple Sequence Alignment (sumber: https://www.biorender.com/template/multiple-sequence-alignment-dna)*

### D. *Needleman-Wunsch Algorithm*

Algoritma *Needleman-Wunsch* merupakan sebuah algoritma dengan pendekatan *dynamic programming*, yang digunakan untuk melakukan alignment antara dua sekuens dan memberikan alignment optimal antara dua sekuens tersebut. Algoritma ini terdiri dari langkah-langkah utama sebagai berikut.

1. Inisialisasi Matriks Skor

   Algoritma ini menggunakan matriks skor seperti pada algoritma alignment sekuens umumnya. Matriks skor ini berukuran $(m+1)\times(n+1)$, dengan $m$ dan $n$ adalah panjang dari kedua sekuens terkait. Setiap sel dalam matriks skor diinisialisasi dengan skor awal berdasarkan aturan substitusi atau matriks substitusi yang telah ditentukan sebelumnya. Pada langkah ini, biasanya terdapat tiga kemungkinan skor yang dapat diberikan:

   - Skor untuk match jika karakter dari kedua sekuens sama.
   - Skor untuk mismatch jika karakter dari kedua sekuens berbeda.
   - Skor untuk gap (celah) jika sebuah karakter harus dihilangkan dari salah satu atau kedua sekuens.

| | | G | C | A | T | G | C | G |
|---|---|---|---|---|---|---|---|---|
| | | | | | | | | |
| G | | | | | | | | |
| A | | | | | | | | |
| T | | | | | | | | |
| T | | | | | | | | |
| A | | | | | | | | |
| C | | | | | | | | |
| A | | | | | | | | |

*Gambar 3 Contoh Inisialisasi Matrix Skor dengan sequence GCATGCG dan GATTACA (sumber: https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm)*

2. Perhitungan Skor Optimal

Setiap sel dalam matriks skor dihitung berdasarkan nilai-nilai sel di sekitarnya. Ini dilakukan dengan menggunakan persamaan rekursif, dimana nilai *M[i][j]* merupakan hasil nilai tertinggi dari antara

- *M[i−1][j−1]+Skor_Match_Mismatch(i,j)*
- *M[i−1][j]+Skor_Gap*
- *M[i][j−1]+Skor_Gap*

dimana *Skor_Match_Mismatch(i,j)* merupakan skor untuk match/mismatch antar karakter dari kedua sekuens dan *Skor_Gap* adalah skor untuk gap. Skor match adalah 1, skor mismatch adalah -1, dan skor gap adalah -1.

Contoh penerapan tahap pengisian sel matriks, misalkan kita ingin mengisi sel matrix yang ditandai dengan huruf X.

| | | G |
|---|---|---|
| | 0 | −1 |
| G | −1 | X |

*Gambar 4 Contoh Matriks sebelum Pengisian Matriks Skor (sumber: https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm)*

Sel tersebut memiliki tiga kandidat nilai:

1. Sel diagonal kiri-atas memiliki nilai 0. Pair G dan G cocok, sehingga nilai skor menjadi 0+1=1
2. Sel atas memiliki nilai -1. Karena bergerak melalui sana melambangkan insert/delete, maka nilai menjadi (-1)+(-1) = -2
3. Sel kiri memiliki nilai -1. Karena bergerak melalui sana melambangkan insert/delete, maka nilai menjadi (-1)+(-1) = -2

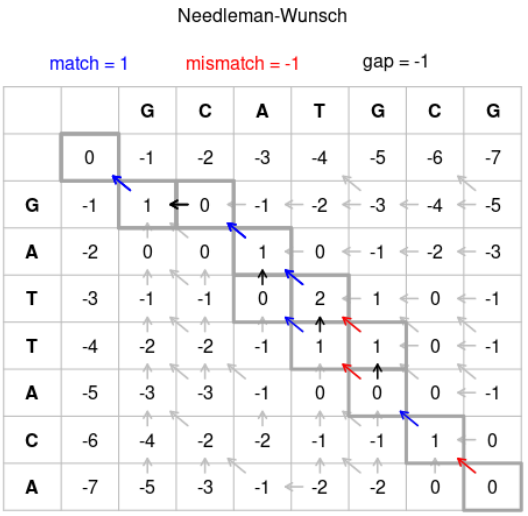
Nilai terbesar terdapat pada kandidat 1, yakni melalui sel diagonal kiri-atas sehingga potongan sel matriks menjadi seperti berikut.

| | | G |
|---|---|---|
| | 0 | −1 |
| G | −1 | 1 |

*Gambar 5 Contoh Matriks setelah Pengisian Matriks Skor (sumber: https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm)*

3. Penentuan Jalur Optimal

Setelah semua nilai sel dalam matriks skor dihitung, jalur optimal yang menghasilkan alignment optimal dapat ditentukan. Jalur optimal akan menghasilkan alignment yang memberikan total skor maksimum.




*Gambar 6 Contoh Ilustrasi Penentuan Jalur Optimal dari skor matriks (sumber:https://en.wikipedia.org/wiki/Needleman%E2%80%93Wunsch_algorithm)*

4. Penghasilan Alignment

Setelah jalur optimal ditentukan, alignment antara kedua sekuens dapat dihasilkan berdasarkan jalur tersebut. Ini melibatkan melacak kembali langkah-langkah yang diambil dalam jalur optimal dan membangun alignment berdasarkan informasi yang ditemukan.

## E. *Algoritma Hirschberg*

Algoritma *Hirschberg* merupakan algoritma *Divide and Conquer* yang digunakan untuk menemukan keselarasan urutan optimal antara dua string. Algoritma ini dapat dipandang sebagai algoritma *Needleman-Wunsch* yang lebih efisien dalam segi waktu dan ruang, yang menggunakan konsep *DnC*.

Algoritma ini memiliki nilai kompleksitas ruang yang lebih efisien dibandingkan algoritma pencocokan sekuens lain seperti *Needleman Wunsch.* Hal ini disebabkan karena pembuatan matriks skor yang hanya mengambil baris akhir saja, sehingga tidak menyimpan seluruh matriks skor seperti pada *Needleman Wunsch*. Hal ini tentu menghemat resource komputasi yang dipakai terutama ketika panjang sekuens-sekuens sangat besar.

Algoritma ini menggunakan paradigma *DnC* dan juga *DP* yang dimana paradigma *DnC* nampak pada pembagian masalah alignment menjadi sub masalah yang lebih kecil, dan paradigma *DP* pada penyelesaian setiap sub masalah tersebut. Langkah-langkah algoritma ini adalah sebagai berikut.

1. *Divide*

   Kedua sekuens yang akan di-align dibagi menjadi dua bagian yang panjangnya ditentukan dari nilai skor matriks *Needleman-Wunsch*. Penentuan batas indeks pembagian ini bertujuan untuk menghasilkan pembagian DnC yang paling optimal. Pembagian ini dilakukan secara rekursif hingga mencapai panjang yang sesuai untuk diproses dengan metode alignment sederhana

2. *Conquer*

   Setelah kedua sekuens dibagi dan mencapai panjang yang sesuai, subsekuens tersebut diselesaikan dengan menggunakan algoritma alignment seperti *Needleman-Wunsch.*

3. *Combine*

   Setelah alignment selesai dilakukan pada subsekuens, solusi dari masing-masing pasangan subsekuens digabungkan untuk menghasilkan solusi alignment global.

## III. Implementasi

Algoritma ini memiliki implementasi seperti algoritma *DnC* umumnya, yang dimana memiliki tahapan *Divide, Conquer,* dan *Combine*. Implementasi fungsi ini akan mengembalikan tuple berisi dua nilai, dimana nilai pertama adalah alignment optimal sekuens pertama, dan nilai kedua adalah alignment optimal sekuens kedua. Implementasi algoritma Hirschberg dilakukan dalam bahasa Python. Berikut adalah penjelasan lebih lanjut terkait implementasi.

1. *Divide*

   Pada tahap ini, sekuens dibagi menjadi subsekuens yang lebih kecil, dimana penentuan panjang subsekuens dari nilai perhitungan baris terakhir dari skor matriks algoritma *Needleman-Wunsch*. Hanya mempertimbangkan baris terakhir matriks ini adalah bentuk optimasi dari *Needleman-Wunsch* yang menghitung keseluruhan matriks.

   Berikut adalah implementasi fungsi pembantu `last_row_NWmtrx` untuk perhitungan skor matrix baris akhir pada algoritma *Needleman Wunsch*.

```
def compute_last_row(seq1, seq2):
    len1 = len(seq1)
    len2 = len(seq2)
    previous_row = list(range(len2 + 1))
    previous_row = [x * -1 for x in
previous_row]
    current_row = [0] * (len2 + 1)

    for i in range(1, len1 + 1):
        current_row[0] = previous_row[0] + -1
        for j in range(1, len2 + 1):
            if seq1[i - 1] == seq2[j - 1]:
                match = previous_row[j - 1] + 1
            else:
                match = previous_row[j - 1] + -1
            delete = previous_row[j] + -1
            insert = current_row[j - 1] + -1
            current_row[j] = max(match, delete,
insert)
        previous_row = current_row[:]

    return current_row
```

*Gambar 7 Implementasi Fungsi count_last_row (sumber: arsip penulis)*

Pencarian baris akhir skor matriks ini berguna untuk menentukan indeks batas pembagian subsekuens kedua sehingga dapat menghasilkan pembagian subsekuens *DnC* yang memastikan skor alignment maksimum dicapai. Sedangkan untuk sekuens pertama, akan dibagi menjadi dua bagian sama panjang. Berikut adalah bagian algoritma *Hirschberg* yang menentukan indeks batas pembagian dan melakukan pemanggilan rekursi untuk hasil pembagian tersebut.

```
def hirschberg(seq1, seq2):
    # ... conquer (basis)

    len1 = len(seq1)
    len2 = len(seq2)
    mid1 = len1 // 2

    upper_score =
compute_last_row(seq1[:mid1], seq2)
    lower_score =
compute_last_row(seq1[mid1:][::-1],
seq2[::-1])

    max_index = 0
    max_score = float('-inf')
    for i in range(len2 + 1):
        score = upper_score[i] +
lower_score[len2 - i]
        if score > max_score:
            max_score = score
            max_index = i

    align1_upper, align2_upper =
hirschberg(seq1[:mid1], seq2[:max_index])
    align1_lower, align2_lower =
hirschberg(seq1[mid1:], seq2[max_index:])

    # ... combine
```

*Gambar 8 Implementasi tahap Divide pada fungsi hirschberg (sumber: arsip penulis)*

2. *Conquer*

   Pada panjang sekuens tertentu, kita kemudian dapat beralih dari pendekatan *DnC* menjadi langsung

   penggunaan algoritma *Needleman-Wunsch* untuk menyelesaikan alignment sekuens. Penentuan batas panjang ini akan menjadi basis dari fungsi *DnC Hirschberg*.

   Penentuan basis pada algoritma ini penting karena terdapat trade-off antara penggunaan memori dan waktu komputasi tergantung dari batas panjang sekuens. Jika terlalu kecill, maka dapat membuat overhead dari rekursi dapat lebih besar dari keuntungan yang didapat. Sebaliknya jika terlalu besar, maka keuntungan pengurangan overhead yang didapat dari *DnC* menjadi sia-sia. Karena algoritma Hirschberg dirancang untuk sekuens-sekuens besar

karena sifat *DnC*-nya, maka penulis menetapkan batas basis panjang sekuens adalah 20.

Basis dari fungsi tersebut adalah ketika salah satu panjang sekuens lebih kecil dari 20, maka akan dilakukan langsung algoritma *Needleman Wunsch.* Untuk basis tambahan, ketika salah satu sekuens memiliki panjang 0, maka akan langsung mengembalikan tuple dengan satu nilai berupa sekuens yang tidak kosong dan nilai lainnya berupa string dengan karakter '-' sepanjang panjang sekuens tidak kosong.

Berikut adalah implementasi dari fungsi *Needleman Wunsch*.

```
def compute_score_mtrx(seq1, seq2):
    len1 = len(seq1)
    len2 = len(seq2)
    matrix = np.zeros((len1 + 1, len2 + 1))

    for i in range(1, len1 + 1):
        matrix[i][0] = matrix[i - 1][0] + -1
    for j in range(1, len2 + 1):
        matrix[0][j] = matrix[0][j - 1] + -1

    for i in range(1, len1 + 1):
        for j in range(1, len2 + 1):
            if seq1[i - 1] == seq2[j - 1]:
                match = matrix[i - 1][j - 1] + 1
            else:
                match = matrix[i - 1][j - 1] +
-1
            delete = matrix[i - 1][j] + -1
            insert = matrix[i][j - 1] + -1
            matrix[i][j] = max(match, delete,
insert)
   return matrix
```

```
def needleman_wunsch(seq1, seq2):
    matrix = compute_score_mtrx(seq1,seq2)

    align1 = ''
    align2 = ''
    i, j = len(seq1), len(seq2)

    while i > 0 and j > 0:
        if seq1[i - 1] == seq2[j - 1] and
matrix[i][j] == matrix[i - 1][j - 1] + 1:
            align1 = seq1[i - 1] + align1
            align2 = seq2[j - 1] + align2
            i -= 1
            j -= 1
        elif matrix[i][j] == matrix[i - 1][j -
1] + -1:
            align1 = seq1[i - 1] + align1
            align2 = seq2[j - 1] + align2
            i -= 1
            j -= 1
        elif matrix[i][j] == matrix[i - 1][j] +
-1:
            align1 = seq1[i - 1] + align1
            align2 = '-' + align2
            i -= 1
        else:
            align1 = '-' + align1
            align2 = seq2[j - 1] + align2
            j -= 1

    while i > 0:
        align1 = seq1[i - 1] + align1
        align2 = '-' + align2
        i -= 1

    while j > 0:
        align1 = '-' + align1
        align2 = seq2[j - 1] + align2
        j -= 1

    return align1, align2
```

*Gambar 9 Implementasi Fungsi needleman_wunsch (sumber: arsip penulis)*

Berikut adalah implementasi basis pada fungsi implementasi algoritma Hirschberg.

```
def hirschberg(seq1, seq2):
    if len(seq1) == 0:
        return '-' * len(seq2), seq2
    elif len(seq2) == 0:
        return seq1, '-' * len(seq1)

    if len(seq1) <= 20 or len(seq2) <= 20:
        return needleman_wunsch(seq1, seq2)

    # ... divide
    # ... combine
```

*Gambar 10 Implementasi tahap Conquer pada Fungsi hirschberg(sumber: arsip penulis)*

*3. Combine*

Setelah mendapatkan alignment optimal untuk kedua pasangan subsekuens, maka tersisa tahap *combine* dimana menggabungkan kedua solusi tersebut menjadi solusi masalah utama. Berikut adalah implementasi *combine* pada algoritma *Hirschberg.*

```
def hirschberg(seq1, seq2):
    # ... conquer (basis)
    # ... divide

    return align1_upper + align1_lower,
align2_upper + align2_lower
```

*Gambar 11 Implementasi tahap Combine pada Fungsi hirschberg (sumber: arsip penulis)*

## IV. Analisis Kasus

A. Contoh Kasus

Pada bagian ini, akan diperjelas tahapan dari algoritma Hirschberg dengan contoh kasus. Dengan adanya contoh kasus, diharapkan pembaca dapat dengan lebih mudah mengerti tahapan implementasi yang dibuat pengguna. Untuk memudahkan contoh kasus, nilai batas basis pada fungsi Hirschberg diubah dari 20 menjadi 3.

Misalkan terdapat dua sekuens genomik sebagai berikut.
*seq1 = AGTACGCA*
*seq2 = TATGC*

Akan dicari indeks pembatas untuk pembagian subsekuens sehingga dapat dipastikan memberikan solusi optimal utama.
Didapatkan nilai skor baris akhir matriks untuk bagian atas dan bawah adalah sebagai berikut.

*upper_score* = [-4, -2, 0, -1, -1, -2]
*lower_score* = [-4, -2, 0, 0, -1, -2]

Akan dilakukan iterasi sesuai implementasi fungsi, hingga akhirnya didapatkan indeks pembatas ideal untuk sekuens kedua adalah indeks 2. Operasi divide kemudian menghasilkan pasangan subsekuens sebagai berikut.
*Rekursi bagian atas*
*hirschberg(seq1[:mid1], seq2[:max_index]) = hirschberg("AGTA", "TA")*

*Rekursi bagian bawah*
*hirschberg(seq1[mid1:], seq2[max_index:]) = hirschberg("CGCA","TGC")*

Proses *Divide* ini dilakukan berulang hingga mencapai basis yakni salah satu sekuens memiliki panjang 0 atau salah satu sekuens memiliki panjang lebih kecil sama dengan 3. Misal kita melanjutkan rekursi bagian atas sebelumnya *hirschberg("AGTA", "TA")*. Dengan mengikuti tahapan pada perhitungan skor matrix pada algoritma *Needleman Wunsch*, didapatkan skor matriks sebagai berikut.

[ 0. -1. -2.]
[-1. -1. 0.]
[-2. -2. -1.]
[-3. -1. -2.]
[-4. -2. 0.]

Setelah mendapatkan skor matriks, dilakukan traceback dari sel matriks kanan bawah hingga sel matriks kiri atas. Traceback dilakukan dengan mengambil nilai terbesar antara sel atas, kiri, dan diagonal atas-kiri, dan kemudian mengaplikasikan gerak tersebut pada sekuens (kiri penyisipan, atas penghapusan, diagonal atas-kiri untuk kesamaan). Setelah dilakukan traceback, didapatkan hasil alignment sebagai berikut.
*optimal alignment "AGTA" = "AGTA"*
*optimal alignment "TA" = "--TA"*

Sedangkan untuk *hirschberg("CGCA","TGC")*, didapatkan hasil optimal alignment sebagai berikut.
*optimal alignment "CGCA" = "CGCA"*
*optimal alignment "TGC" = "TGC-"*

Kemudian dilakukan operasi *combine*, sehingga didapatkan solusi optimal masalah utama dengan sekuens AGTACGCA dan TATGC adalah
*("AGTA"+"CGCA", "--TA"+"TGC-")*
*= ("AGTACGCA","--TATGC-")*

Berikut adalah hasil *screenshot* ketika dijalankan dalam program.

```
PS C:\Users\user\Documents\Hirschberg> python Hirschberg.py
Masukkan sekuens genomik pertama: AGTACGCA
Masukkan sekuens genomik kedua: TATGC
didapat hasil alignment optimal dari kedua sekuens adalah
alignment optimal sekuens genomik pertama "AGTACGCA": AGTACGCA
alignment optimal sekuens genomik kedua "TATGC": --TATGC-
```

*Gambar 12 Hasil Alignment Optimal Sequence dengan input AGTACGCA dan TATGC(sumber: arsip penulis)*

B. Analisis Kompleksitas

- Kompleksitas Waktu

  Algoritma Hirschberg memiliki kompleksitas waktu *O(mn)*, di mana *m* dan *n* adalah panjang dari dua urutan yang dibandingkan. Ini didapat dari proses *Divide* yang secara rekursif membagi menjadi dua subsekuens pada tiap iterasi fungsi. Sedangkan algoritma Needleman Wunsch memiliki kompleksitas waktu *O(mn)* juga karena algoritma harus mengisi seluruh matriks skor berukuran $m \times n$.

  Dapat dilihat bahwa dari segi kompleksitas waktu, tidak ada peningkatan efisiensi signifikan dari algoritma Hirschberg dibandingkan algoritma *Needleman Wunsch*.

- Kompleksitas Ruang

  Algoritma Hirschberg memiliki kompleksitas ruang *O(m+n)*, karena pada setiap langkah rekursif, hanya diperlukan penyimpanan sementara untuk satu baris matriks dari matriks skor *Needleman Wunsch*. Sedangkan algoritma *Needleman Wunsch* memiliki kompleksitas ruang *O(mn)* karena algoritma ini harus menyimpan seluruh matriks $m \times n$ untuk melacak skor pencocokan antara semua pasangan dari dua sekuens.

  Dapat dilihat bahwa dari segi kompleksitas ruang, terdapat peningkatan efisiensi ruang pada algoritma Hirschberg karena hanya menyimpan baris akhir dari matriks, sedangkan *Needleman Wunsch* menyimpan keseluruhan matriks. Efisiensi ruang ini berguna ketika sekuens genom menjadi sangat panjang, sehingga akan menghemat ruang yang secara tidak langsung akan berpengaruh pada performa program.

## V. KESIMPULAN

Pencocokan sekuens genomik merupakan aspek yang fundamental dalam membandingkan dan menganalisis urutan DNA atau RNA dari berbagai organisme. Hal ini dapat diselesaikan dengan penerapan DP menggunakan algoritma *Needleman Wunsch* untuk menentukan alignment optimal yang dapat dengan mudah dilakukan pencocokan sekuens. Akan tetapi, untuk sekuens genomik yang sangat panjang

seperti pada organisme-organisme di dunia nyata, algoritma ini akan memiliki penurunan performa karena nilai kompleksitas ruang yang besar ($O(m \times n)$).

Permasalahan kompleksitas ruang ini dapat dioptimasi dengan menerapkan konsep *Divide and Conquer* berupa algoritma Hirschberg. Dengan menerapkan konsep *divide and conquer* ini, kompleksitas ruang dapat diperingan hingga menjadi $O(m+n)$. Hal ini sangat berguna terutama untuk sekuens-sekuens yang sangat panjang. Dengan kompleksitas ruang yang lebih efisien, performa program akan meningkat ketika melakukan perhitungan sekuens alignment pada sekuens-sekuens dengan jumlah panjang huruf yang sangat besar seperti pada genom populasi.

## VI. Ucapan Terima Kasih

Penulis ingin mengucapkan terima kasih kepada semua dosen IF2211 Strategi Algoritma, khususnya kepada Dr. Ir. Rinaldi, M.T sebagai dosen pengajar di Kelas 1 IF2210 untuk Strategi Algoritma, atas pengajarannya dan dukungannya pada para mahasiswa/i untuk menulis makalah ini. Melalui makalah ini, saya semakin mengerti dan paham tentang algoritma *Divide and Conquer* dan algoritma *Dynamic Programming*. Saya juga ingin berterima kasih kepada Dr. Ir. Rinaldi, M.T, yang telah menyediakan media pembelajaran bagi para mahasiswa/i

## References

[1] Needleman, Saul B. & Wunsch, Christian D. (1970). "A general method applicable to the search for similarities in the amino acid sequence of two proteins". Journal of Molecular Biology. 48 (3): 443–53. doi:10.1016/0022-2836(70)90057-4. PMID 5420325.

[2] Mount DM. (2004). Bioinformatics: Sequence and Genome Analysis (2nd ed.). Cold Spring Harbor Laboratory Press: Cold Spring Harbor, NY. ISBN 978-0-87969-608-5.

[3] Gagniuc, Paul A. (2021). Algorithms in bioinformatics : theory and implementation (1st ed.). Hoboken, NJ: John Wiley & Sons. ISBN 978-1-119-69800-5. OCLC 1240827446.

[4] R. Munir, "Matematika Diskrit - ITB, 2023-2024," Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung. [Online]. Available: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2023-2024/stima23-24.htm. Diakses pada Juni 11, 2023.

[5] Lloyd Allison. "The Hirsch Conjecture," Monash University, School of Computer Science and Software Engineering. [Online]. Available: https://www.csse.monash.edu.au/~lloyd/tildeAlgDS/Dynamic/Hirsch/. Diakses pada Juni 11, 2024.

## PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 12 Juni 2024

Ignatius Jhon Hezkiel Chan
13522029