Pencarian Periode Investasi Saham Paling Menguntungkan Menggunakan Divide and Conquer (Maximum Subarray Problem)

Hasri Fayadh Muqaffa - 13523156
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: hasri.fayadh@gmail.com , 13523156@std.stei.itb.ac.id

***Abstrak*—Makalah ini bertujuan untuk mengidentifikasi periode investasi tunggal (satu kali beli atau satu kali jual) yang paling menguntungkan berdasarkan data historis harga saham dengan menerapkan paradigma algoritma *Divide and Conquer*. Penentuan waktu transaksi yang optimal (*market timing*) merupakan salah satu masalah fundamental di dalam dunia investasi. Makalah ini akan mentrasformasikan permasalahan tersebut menjadi *Maximum Subarray Problem* (MSP). Transformasi ini dapat dilakukan dengan mengubah data deret waktu harga penutupan saham menjadi sebuah *array* yang berisi perubahan harga harian. Algoritma *Divide and Conquer* akan diimplementasikan untuk menyelesaikan MSP ini melalui tiga Langkah, yaitu *divide* (membagi) *array* data menjadi sub-masalah yang lebih kecil, *conquer* (menyelesaikan) sub-masalah secara rekursif, dan *combine* (menggabungkan) hasilnya untuk menemukan solusi. Pada makalah ini, data historis saham PT Bank Central Asia Tbk (BBCA.JK) digunakan sebagai bahan percobaan dan berhasil mengidentifikasi periode jual-beli dengan keuntungan maksimal. Meskipun efektif sebagai alat analisis historis, model ini memiliki keterbatasan untuk aplikasi praktis karena bergantung pada informasi masa lalu dan mengabaikan faktor dunia nyata yang sedang terjadi.**

Keywords—Investasi, waktu transaksi yang optimal, maximum subarray problem, divide and conquer, data historis

PENDAHULUAN

Investasi di pasar saham merupakan kegiatan keuangan yang dapat memberikan potensi keuntungan (*return*) yang besar, tetapi diiringin juga dengan risiko (*risk*) yang sepadan. Salah satu tantangan terbesar yang dihadapi oleh investor adalah *market timing*, yaitu strategi untuk memutuskan kapan waktu yang tepat untuk membeli atau menjual asset.[1] Tujuan dari *market timing* adalah untuk membeli saham pada harga rendah dan menjual pada harga tinggi, sehingga dapat memaksimalkan profit (*return*). Akan tetapi, strategi ini sangat sulit untuk dilakukan secara konsisten dan akurat. Pasar saham bersifat sangat fluktuatif dan sulit untuk diprediksi, dipengaruhi oleh berbagai faktor ekonomi, poltik, dan juga sentimen pasar yang kompleks.

Banyak ahli keuangan yang merekomendasikan untuk melakukan investasi dengan pendekatan jangka panjang, seperti strategi *buy and hold* yang lebih berfokus pada nilai fundamental saham daripada mencoba memprediksi fluktuasi pasar secara jangka pendek. Meskipun demikian, banyak orang yang masih tertarik untuk dapat mengidentifikasi periode paling menguntungkan secara historis, sehingga topik ini masih relevan. Analisis ini dapat berfungsi sebagai tolak ukur untuk mengevaluasi kinerja strategi investasi yang sebenarnya dan memahami dinamika pasar pada periode tertentu.

Dengan kemajuan teknologi komputasi, analisis data historis dalam skala besar menjadi lebih mudah untuk dilakukan. Algoritma di bidang keinformatikaan dapat diterapkan untuk menganalisis data harga saham dan menemukan pola atau solusi optimal untuk masalah yang terdefinisi dengan baik.

Masalah inti yang akan dipecahkan dalam makalah ini adalah bagaimana menentukan satu hari optimal untuk membeli dan satu hari optimal untuk menjual yang akan menghasilkan keuntungan maksimal jika diberikan serangkain data harga saham historis untuk satu periode. Kunci untuk menyelesaikan masalah ini secara efektif terletak pada transformasi data atau abstraksi. Misalkan dimiliki serangkaian harga penutupan harian $P = [P_0, P_1, P_2, \dots P_n]$. Keuntungan dari membeli pada hari ke-i dan menjual pada hari ke-j (dengan j > i) adalah:

$$Profit = P_j - P_i$$

Selisih ini dapat diekspresikan sebagai jumlah dari semua perubahan harga harian dari i + 1 hingga hari j:

$$P_j - P_i = (P_j - P_{j-1}) + (P_{j-1} - P_{j-2}) + \cdots + (P_{i+1} - P_i)$$

Jika didefinsikan sebuah *array* baru, ΔP, yang berisi perubahan harga harian dengan P_{k-1} untuk $k = 1, \dots, n$, maka keuntungan di atas menjadi:

$$Profit = \sum_{k=i+1}^{j} \Delta P_k$$

Dengan melakukan transformasi, masalah mencari keuntungan maksimal dari transaksi saham tunggal dapat

berubah menjadi masalah menemukan *subarray* bersebelahan (*contiguous subarray*) dalam *array* perubahan harga. Dalam bidang keinformatikaan, ini disebut dengan *maximum subarray problem*. Makalah ini akan berfokus untuk menyelesaikan permasalahan MSP ini dengan menggunakan paradigma algoritma *divide and conquer*.

Berdasarkan penjelasan sebelumnya, dapat ditetapkan empat tujuan dalam pembuatan makalah ini, yaitu melakukan implementasi algoritma *Divide and Conquer* untuk menyelesaikan *Maximum Subarray Problem*, menerapkan algoritma yang telah diimplementasikan pada data historis saham nyata dengan melakukan percobaan pada saham PT Bank Central Asia Tbk (BBCA.JK), menganalisis kinerja dan kompleksitas komputasi dari algoritma *Divide and Conquer* dalam konteks penyelesaian masalah ini, dan mengevaluasi secara kritis asumsi-asumsi yang mendasari model ini serta mengidentifikasi keterbasan untuk digunakan secara langsung di dunia nyata.

Adapun ruang lingkup dari makalah ini adalah analisis hanya dilakukan pada data historis dari satu emiten saham, yaitu BBCA.JK, model yang dikembangkan mengasumsikan bahwa investor hanya melakukan satu kali transaksi pembelian dan satu kali transaksi penjualan, faktor-faktor nyata di pasar dunia diabaikan, seperti biaya transaksi, pajak atas keuntungan modal, dan likuiditas pasar, dan algoritma ini beroperasi dengan asumsi *perfect hindsight*, yaitu memiliki akses penuh ke seluruh data historis.

DASAR TEORI

A. *Investasi Saham dan Konsep Market Timing*

Investasi saham merupakan salah satu instrument investasi yang paling popular, yaitu investor membeli Sebagian kecil kepemilikan dalam sebuah Perusahaan publik dengan harapan nilai saham tersebut akan meningkat di masa depan. Strategi investasi secara umum dapat dibagi menjadi dua pendekatan utama, yaitu investasi berbasis nilai (*fundamental*) dan berbasis *market timing* (*technical*).

Strategi *buy and hold* merupakan salah satu strategi dari pendekatan berbasis nilai dengan menganjurkan investor untuk membeli saham dari Perusahaan-perusahaan yang fundamentalnya kuat dan dapat menahannya untuk jangka waktu yang lama, terlepas dari fluktuasi pasar jangka pendek.[2] Di sisi lain, *market timing* adalah pendekatan yang lebih aktif dan spekulatif. Investor yang menggunakan strategi ini mencoba untuk memprediksi pergerakan pasar untuk membeli saham sebelum harganya naik dan menjualnya sebelum harganya turun. Keuntungan potensial dari *market timing* yang berhasil adalah sangat besar karena memungkinkan investor untuk mendapatkan keuntungan dari volatilitas pasar. Namun, strategi ini juga memiliki risiko yang sangat tinggi. Beberapa risiko utamanya adalah kesulitan untuk memprediksi harga pasar (pasar sangat kompleks dan dipengaruhi oleh banyak variabel yang tidak dapat diprediksi), kehilangan peluang (*opportunity loss*) (kesalahan dalam memprediksi dapat menyebabkan investor menjual terlalu dini dan kehilangan potensi keuntungan lebih lanjut), dan biaya transaksi yang tinggi (*market timing* seringkali melibatkan frekuensi transaksi yang tinggi). Meskipun berisiko, analisis *market timing* secara historis tetap menjadi alat yang berguna. Dengan menganalisis data historis, kita dapat mengidentifikasi periode "emas" yang dapat memberikan hasil maksimal.

B. *Maximum Subarray Problem (MSP)*

Maximum Subarray Problem (MSP) adalah salah satu masalah fundamental dalam ilmu komputer. Definisi formal dari masalah ini adalah sebagai berikut:

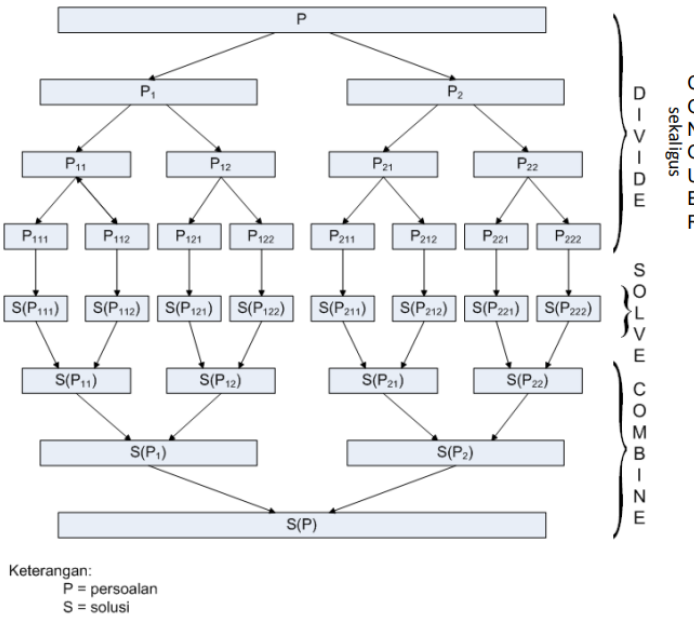
- Diberikan sebuah *array* satu dimensi A yang berisi n bilangan, temukan sebuah *subarray* bersebelahan (*contiguous subarray*) $A[i \ldots j]$ (dengan $0 \leq i \leq j \leq n$) yang jumlah elemen-elemennya adalah yang terbesar.[3]
- Sebagai contoh, jika diberikan *array* $A = [-2, 1, -3, 4, -1, 2, 1, -5, 4]$, maka *subarray* dengan jumlah terbesar adalah $[4, -1, 2, 1]$ yang totalnya adalah 6.

Masalah ini memiliki beberapa beberapa kasus khusus yang menarik[4], yaitu:

- Jika semua elemen positif, solusinya trivial, yaitu seluruh *array* itu sendiri
- Jika semua elemen negative, solusinya adalah *subarray* yang hanya berisi satu elemen dengan nilai terbesar
- Jika *array* kosong diperbolehkan sebagai solusi dan jumlahnya didefinisikan sebagai 0, maka jika semua elemen dalam *array* adalah negatif, solusi optimalnya adalah *subarray* kosong dengan jumlah 0. Akan tetapi, pada permasalahan kali ini, kita mencari *subarray* yang tidak kosong

C. *Paradigma Algoritma Divide and Conquer (DnC)*

Algoritma *Divide and Conquer* adalah sebauh sebuah desain algoritma yang didasarkan pada rekursi. Ide dasarnya adalah memecahkan masalah yang besar dan kompleks dengan cara membaginya menjadi beberapa sub-masalah yang lebih kecil dan lebih mudah untuk diselesaikan. Sub-masalah ini harus memiliki tipe yang sama dengan masalah asli. Algoritma DnC ini terdiri dari tiga langkah utama[5], yaitu:



Gambar 1.
Tahapan Algoritma Divide and Conquer diambil dari
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf

1. *Divide* (membagi). Langkah ini melibatkan pemecahan masalah utama menjadi dua atau lebih sub-masalah yang lebih kecil. Idealnya, setiap sub-masalah memiliki ukuran yang hampir sama. Proses pembagian ini dilakukan secara rekursif hingga sub-masalah menjadi cukup sederhana untuk diselesaikan secara langsung.
2. *Conquer* (menyelesaikan). Pada langkah ini, setiap sub-masalah diselesaikan. Jika sub-masalah sudah mencapai ukuran minimal (*base case*), solusinya dihitung scara langsung. Jika sub-masalah masih cukup besar, akan diterapkan Kembali secara rekursif pada sub-masalah tersebut.
3. *Combine* (menggabungkan). Setelah semua sub-masalah diselesaikan, solusi dari masing-masing sub-masalah tersebut digabungkan untuk membentuk solusi dari masalah asli.

Keunggulan utama dari *Divide and Conquer* adalah kemampuan untuk menghasilkan algoritma yang sangat efisien.

D. Penerapan Divide and Conquer pada Maximum Subarray Problem

Penerapan paradigma *Divide and Conquer* pada *Maximum Subarray Problem* merupakan contoh klasik dalam ilmu komputer. Contohnya adalah misalnya ingin dicari *maximum subarray* dari *array* A pada rentang indeks low hingga high. Maka, Langkah-langkahnya sebagai berikut:

1. *Divide*: Langkah ini akan mencari indeks tengah dari *array*, yaitu $mid = floor(\frac{(low+high)}{2})$. Indeks ini membagi *array* $A[low \ldots high]$ menjadi dua *subarray*, yaitu *subarray* kiri $A[low \ldots mid]$ dan *subarray* kanan $A[mid + 1 \ldots high]$.
2. *Conquer*: Pada langkah ini, secara rekursif akan memanggil fungsi untuk menyelesaikan MSP pada kedua *subarray* tersebut. Rekursif akan terus berlanjut hingga mencapai *base case*, yaitu ketika *subarray* hanya memiliki satu elemen. Dalam kasus ini, *maximum subarray* adalah elemen itu sendiri.
3. *Combine*: Ini merupakah langkah terakhir. Setelah mengetahui solusi dari *subarray* kiri dan *subarray* kanan, setiap penyelesaian dapat digabungkan untuk menemukan solusi dari *array* asli (*array* awal). *Maximum subarray* dari $A[low \ldots mid]$ pasti berada di salah satu dari tiga kemungkinan berikut:
    a. Sepenuhnya berada di dalam *subarray* kiri, yaitu *subarray* $A[i \ldots j]$ dengan $low \leq i \leq j \leq mid$.
    b. Sepenuhnya berada di dalam *subarray* kanan, yaitu *subarray* $A[i \ldots j]$ dengan $mid + 1 \leq i \leq j \leq high$.
    c. Melintasi titik tengah, yaitu *subarray* $[i \ldots j]$ dengan $low \leq i \leq mid < j \leq high$. Untuk kasus ini, perlu dicari *maximum subarray* yang dimulai dari suatu indeks i di sebelah kiri mid dan berakhir di mid, lalu menambahkannya dengan *maximum subarray* yang dimulai dari mid + 1 dan berakhir di suatu indeks j di sebelah kanan mid + 1. Proses ini dapat dilakukan dengan dua iterasi linear dari titik tengah ke arah luar.

Solusi akhir dari masalah ini adalah nilai maksimum dari ketiga kasus di atas. Kompleksitas dari langkah *combine* adalah linear.

IMPLEMENTASI DAN PENGUJIAN

A. Deskripsi dan Akuisisi Data

Untuk menerapkan dan menguji algoritma, diperlukan data historis harga saham yang akurat dan relevan. Dalam makalah ini, data yang digunakan adalah data harga saham harian dari PT Bank Central Asia Tbk, salah satu Perusahaan dengan kapitalisasi pasar tersebsar di Bursa Efek Indonesia (BEI) dengan kode emiten BBCA.JK. Sumber data historis didapatkan dari Kaggle[6]. Data yang diambil mencakup dari 1 Januari 2019 hingga 17 Februari 2025. Data diunduh dalam format *Comma-Separated Values* (CSV). Setiap baris dalam file CSV merepresentasikan data pergadangan untuk satu hari dengan penjelasan kolom-kolomnya sebagai beikut:

- Date: Tanggal pencatatan data
- Open: Harga pembukaan saham pada hari tersebut

- High: Harga tertinggi yang dicapai pada hari tersebut
- Low: Harga terendah yang dicapai pada hari tersebut
- Close: Harga penutupan saham pada hari tersebut
- Adj Close: Harga penutupan yang telah disesuaikan untuk aksi korporasi, seperti dividen dan *stock split*
- Volume: Jumlah saham yang diperdagangkan pada hari tersebut

Untuk analisis dalam makalah ini, kolom yang akan menjadi fokus utama adalah harga penutupan (Close) karena secara umum merepresentasikan nilai saham pada akhir hari perdagangan.

B. Pra-pemrosesan Data dengan Python

Sebelum data dapat digunakan oleh algoritma, perlu dilakukan tahap pra-pemrosesan. Tahap ini bertujuan untuk membersihkan dan mentransformasikan data mentah menjadi format yang sesuai untuk *Maximum Subarray Problem*. Proses ini dilakukan menggunakan Bahasa pemrograman Python dengan *libraru* pandas. Langkah-langkahnya sebagai berikut:

1. Pertama, file csv yang berisi data historis BBCA.JK akan dibaca dan dimuat ke dalam sebuah struktur data yang disebut DataFrame pandas. Kolom 'Date' diatur sebagai indeks untuk mempermudah analisis berdasarkan waktu
2. Setelah itu, akan dilakukan pengubahan data harga absolut menjadi data perubahan harga harian.
3. Hasil dari langkah kedua berupa list yang berisi fluktuasi harga harian. List ini lah yang akan menjadi input utama untuk algoritma *Divide and Conquer*.

C. Implementasi Algoritma Divide and Conquer

Implementasi ini terdiri dari dua fungsi utama, yaitu max_crossing_sum untuk langkah *combine* dan find_maximum_subarray sebagai fungsi rekursif utama.

1. Fungsi max_crossing_sum berfungsi untuk mencari *maximum subarray* yang melintasi titik tengah (mid). Fungsi ini bekerja dengan melakukan dua pemindaian, yaitu satu dari mid ke kiri (low) dan satu dari mid + 1 ke kanan (high)

```python
def max_crossing_sum(arr, low, mid, high):
    # Inisialisasi variabel untuk pencarian di sisi kiri dari titik tengah
    left_sum = float('-inf')
    current_sum = 0
    max_left_index = mid

    # Iterasi dari titik tengah ke arah kiri
    for i in range(mid, low - 1, -1):
        current_sum += arr[i]
        if current_sum > left_sum:
            left_sum = current_sum
            max_left_index = i

    # Inisialisasi variabel untuk pencarian di sisi kanan dari titik tengah
    right_sum = float('-inf')
    current_sum = 0
    max_right_index = mid + 1

    # Iterasi dari titik tengah ke arah kanan
    for i in range(mid + 1, high + 1):
        current_sum += arr[i]
        if current_sum > right_sum:
            right_sum = current_sum
            max_right_index = i

    # Mengembalikan indeks dan jumlah total dari subarray yang melintasi tengah
    return (max_left_index, max_right_index, left_sum + right_sum)
```

Gambar 2. Implementasi dari Fungsi max_crossing_sum diambil dari VS Code

2. Fungsi find_maximum_subarray berfungsi untuk mengimplementasikan logika *Divide and Conquer* secara rekursif.
3.

```python
def find_maximum_subarray(arr, low, high):
    # Base Case: Jika array hanya memiliki satu elemen
    if low == high:
        return (low, high, arr[low])
    else:
        # Divide: Menentukan titik tengah untuk membagi array
        mid = (low + high) // 2

        # Conquer: Melakukan rekursif untuk sisi kiri dan kanan
        left_low, left_high, left_sum = find_maximum_subarray(arr, low, mid)
        right_low, right_high, right_sum = find_maximum_subarray(arr, mid + 1, high)

        # Combine: Mencari maximum subarray yang melintasi titik tengah
        cross_low, cross_high, cross_sum = max_crossing_sum(arr, low, mid, high)

        # membandingkan ketiga hasil (kiri, kanan, melintas titik tengah)
        if left_sum >= right_sum and left_sum >= cross_sum:
            return (left_low, left_high, left_sum)
        elif right_sum >= left_sum and right_sum >= cross_sum:
            return (right_low, right_high, right_sum)
        else:
            return (cross_low, cross_high, cross_sum)
```

Gambar 3. Implementasi Fungsi find_maximum_subbaray diambil dari VS Code

D. Pengujian

Setelah data diproses dan algoritma diimplementasikan, langkah selanjutnya adalah melakukan pengujian pada data saham BBCK.JK. Berikut gambar pengujian

```
--- Hasil Analisis Periode Investasi Optimal ---
Periode Data: 2019-01-01 hingga 2025-02-17
------------------------------------------------
Tanggal Beli Optimal : 2020-03-23
Harga Beli           : Rp 4,430.00
Tanggal Jual Optimal : 2024-09-23
Harga Jual           : Rp 10,950.00
Keuntungan Maksimal  : Rp 6,520.00 per lembar saham
------------------------------------------------
```

Gambar 4. Hasil Pengujian diambil dari Terminal VS Code

Hasil yang ditampilkan dari eksekusi kode di atas adalah tanggal beli dan jual yang spesifik dan juga harga pada saat itu serta total keuntungan yang akan didapatkan.

ANALISIS

A. *Analisis Hasil Implementasi*

Setelah melakukan pengujian algoritma menggunakan data historis saham BBCA.JK dari tanggal 1 Januari 2019 hingga 17 Februari 2025 diperoleh hasil periode investasi Tunggal yang paling menguntungkan. Interprestasi dari hasil tersebut adalah dalam rentang waktu yang telah dianalisis, strategi *buy and hold* yang paling untung adalah dengan membeli saham BBCA pada penutupan pasar tanggal 23 Maret 2020 dan menjualnya pada tanggal 23 September 2024. Periode ini memberikan keuntungan tertinggi dibandingkan kombinasi tanggal beli-jual lainnya.

B. *Analisis Kompleksitas Algoritma*

1. Kompleksitas Waktu

   Kompleksitas waktu dari algoritma *Divide and Conquer* untuk *Maximum Subarray Problem* adalah $O(n \log n)$. Kompleksitas ini dapat diturunkan dengan menganalisis relasi rekurensi dari algoritma. Misalkan $T(n)$ adalah waktu yang dibutuhkan untuk menjalankan algoritma pada *array* berukuran n, maka:

   - *Divide*: Langkah ini akan membagi *array* menjadi dua bagian, sehingga hanya membutuhkan waktu konstan, $O(1)$.
   - *Conquer*: Langkah ini melakukan dua rekursif pada *subarray* yang berukuran setengah dari ukuran asli, yaitu n/2. Waktu yang dibutuhkan adalah $2T(\frac{n}{2})$.
   - *Combine*: Langkah ini melakukan penggabungan dengan menggunakan fungsi max_crossing_sum dengan kompleksitas waktu $O(n)$.

   Dengan menggabungkan semua langkah di atas, relasi rekurensi untuk algoritma ini adalah $T(n) = 2T\left(\frac{n}{2}\right) + O(n)$ yang mana ini idenyik dengan algoritma *Merge Sort*. Dengan menggunakan *Master Theorem*, didapatkan solusi untuk relasi ini adalah $T(n) = O(n \log n)$.

2. Kompleksitas Ruang

   Kompleksitas ruang dari algoritma ini adalah $O(\log n)$. Ruang ini digunakan oleh tumpukan panggilan rekursif (*recursion call stack*). Karena di setiap langkah *array* dibagi dua, kedalaman maksimum dari pohon rekursi adalah $log_2 n$. Setiap pemanggilan, fungsi menyimpan beberapa variabel lokal, seperti low, high, mid, dan hasil dari sub-masalah yang membutuhkan ruang konstan. Oleh karean itu, total ruang yang dibutuhkan sebanding dengan kedalaman rekursi, yaitu $O(\log n)$

3. Analisis Perbandingan Algoritma

   Meskipun pendekatan *Divide and Conquer* efisien, perlu untuk dilakukan perbandingan dengan algorima lainnya. Ini dilakukan untuk memahami *trade-off* antara berbagai pendekatan. Terdapat tiga pendekatan utama untuk permasalahan ini, yaitu

   a. Pendekatan *Brute-Force*. Pendekatan ini merupakan pendekatan yang paling intuitif. Algoritma ini menggunakan dua *loop* bersarang untuk menghasilkan semua kemungkinan *subarray*, menghitung jumlah, dan melacak nlai maksimum yang ditemukan. Kompleksitas waktu dari pendekatan ini adalah $O(n^2)$.

   b. Pendekatan *Divide and Conquer*. Pendekatan ini memecah masalah secara rekursif dan memiliki kompleksitas waktu $O(n \log n)$.

   c. Algoritma Kadane (*Dynamic Programming/Greedy*). Pendekatan ini adalah solusi yang paling optimal untuk MSP. Algoritma ini melakukan satu kali pemindaian (*single pass*) pada *array* sambil mempertahankan jumlah *subarray* maksimum yang berakhir pada posisi saat ini. Algoritma ini sangat efisien dengan kompleksitas waktu linear $O(n)$ dam kompleksitas ruang konstan $O(1)$.

   Berdasarkan penjelasan di atas, meskipun *Divide and Conquer* memiliki peningkatan yang besar dari *Brute-Force*, tetapi algoritma Kadane merupakan solusi terbaik dari segi efisiensi.

C. *Analisis Kritis dan Keterbatasan Model*

Meskipun algoritma berhasil menemukan solusi matematis untuk masalah ini, perlu diperhatikan untuk melakukan analisis kritis terhadap model ini dan memahami keterbeasan dalam konteks untuk digunakan langsung dalam dunia nyata. Adapun beberapa keterbatasannya adalah sebagai berikut:

1. Model ini bergantung pada informasi yang harus lengkap (dalam pengujian ini, digunakan data lengkap historis dari emiten BBCA). Di dunia nyata, Keputusan investasi harus dibuat dengan informasi yang tidak lengkap dan ketidakpastian mengenai kedepannya.
2. Model ini diasumsikan hanya untuk transaksi tunggal. Dalam dunia nyata, banyak orang yang melakukan transaksi secara berulang. Masalah transaksi berulang atau beberapa transaksi (*multiple transactions*) merupakan masalah yang berbeda dan jauh lebih kompleks, sehingga memerlukan pendekatan algoritma yang berbeda juga.
3. Model ini beroperasi tanpa ada biaya, seperti biaya komisi, *bid-ask spread, slippage*, dan juga pajak.

KESIMPULAN

Makalah ini telah berhasil menunjukkan penerapan paradigma algoritma *Divide and Conquer* untuk menyelesaikan masalah penentuan periode investasi saham tunggal yang paling menguntungkan. Melalui transformasi data harga historis menjadi *array* perubahan harga harian, masalah investasi ini berhasil dipetakan menjadi *Maximum Subarray Problem* (MSP).

Implementasi algoritma DnC pada data saham PT Bank Central Asia Tbk (BBCA.JK) berhasil mengidentifikasi secara akurat tanggal beli dan tanggal jual yang menghasilkan keuntungan maksimal dalam periode waktu yang dianalisis. Analisis kompleksitas menunjukkan bahwa algoritma ini memiliki kompleksitas waktu $O(n \log n)$ dan kompleksitas ruang $O(\log n)$ yang lebih efisien jika dibandingkan dengan *Brute-Force* yang memiliki kompleksitas $O(n^2)$.

Akan tetapi, perlu digarisbawahi bahwa model yang telah dibuat ini memiliki keterbatasan. Meskipun secara matematis valid, model ini berfungsi sebagai alat analisis historis dan *benchmark* teoretis, bukan sebagai aplikasi langsung di dunia nyata.

REKOMENDASI UNTUK PENGEMBANGAN LEBIH LANJUT

Berdasarkan analisis dan keterbatasan yang telah disebutkan sebelumnya, berikut terdapat beberapa rekomendasi untuk pengembangan lanjutannya, yaitu:

1. Mengimplementasikan algoritma yang lebih optimal, yaitu algoritma Kadane. Algoritma ini dapat menyelesaikan MSP dalam waktu linear dan merupakan solusi yang paling efisien.
2. Untuk membuat model yang lebih realistis, dapat memodifikasi fungsi perhitungan profit dengan memasukkan berbagai jenis biaya transaksi.
3. Mengganti *hindsight* (informasi sempurna secara historis) dengan model prediktif. Ini merupakan lompatan agara dapat digunakan secara praktis dalam dunia nyata. Pengembangan selanjutnya dapat berfokus pada penggunaan *time-series forecasting*, seperti ARIMA (*AutoRegresive Integrated Moving Average*) atau model *Machine Learning* yang lebih canggih, seperti *Long Short-Term Memory* (LSTM) *Networks*.
4. Generalisasi untuk transaksi ganda (*multiple transactions*). Pengembangan selanjutnya mungkin dapat melakukan eksplorasi algoritma yang dirancang untuk masalah *Stock Buy Sell with Multiple Transactions*. Masalah ini memungkinkan untuk melakukan serangkaian transaksi beli dan jual untuk memaksimalkan total akumulasi keuntungan. Masalah ini memerlukan pendekatan *Dynamic Programming*.

LINK VIDEO YOUTUBE

https://www.youtube.com/watch?v=4UxasXv_YNQ

UCAPAN TERIMA KASIH

Penulis ingin menyampaikan rasa terima kasih kepada Tuhan yang Maha Esa atas berkat, rahmat, dan kasih sayangnya-Nya yang telah memungkinkan penulis dapat menyelesaikan masalah ini tanpa ada kendala dan tantangan yang terlalu berat. Penghargaan juga disampaikan kepada keluarga, kerabat, dan teman-teman yang telah memberikan dukungan sehingga penulis berhasil menyelesaikan makalah.

Tak lupa, penulis ingin mengucapkan terima kasih kepada para dosen pengampuh mata kuliah IF2211, khususnya kepada pak Monterico Adrian, S.T., M.T. yang telah memberikan pengetahun dan pembelajaran yang sangat berharga bagi penulis selama perkuliahan. Penulis berharap makalah ini dapat dimanfaatkan dengan bagus, baik untuk penulis, maupun orang lain.

REFERENSI

[1] https://www.bizhare.id/media/investasi/market-timing diakses pada 22 Juni 2025

[2] https://www.mikirduit.com/strategi-investasi-saham-terbaik-market-timing-vs-buy-and-hold/ diakses pada 22 Juni 2025

[3] https://algo.monster/liteproblems/53 diakses pada 23 Juni 2025

[4] https://blog.heycoach.in/maximum-subarray-sum-problem/ diakses pada 23 Juni 2025

[5] https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf diakses pada 23 Juni 2025

[6] https://www.kaggle.com/datasets/caesarmario/bank-central-asia-stock-historical-price diakses pada 23 Juni 2025

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, 24 Juni 2025

Hasri Fayadh Muqaffa dan 13523156