Penyelesaian Studi Kasus Pencarian *Royal Flush* dalam Suatu *Deck* Kartu Poker yang Terurut menggunakan Metode *Sequential Search* dan *Binary Search*

Diajukan untuk memenuhi salah satu tugas mata kuliah CII2K3 - Strategi Algoritma

Disusun oleh:

Ryan Oktaviandi Susilo Wibowo	**1301204289**
Muhammad Khalid Habiburahman	**1301204437**
Raihan Atsal Hafizh	**1301204485**

Kelompok 1
IF-44-09

PROGRAM STUDI S1 INFORMATIKA
FAKULTAS INFORMATIKA
TELKOM UNIVERSITY

2022

Daftar Isi

ABSTRAK

Penelitian ini bertujuan untuk menguji dan membandingkan keefektifan dan keefisienan antara metode *brute force* dan *divide and conquer* untuk memecahkan suatu masalah permainan poker agar menghasilkan kartu *royal flush* pada *deck* kartu poker yang baru. Analisis ini menggunakan algoritma *sequential search* sebagai metode *brute force* dan algoritma *binary search* sebagai metode *divide and conquer*. Penelitian ini dilakukan melalui tiga tahap utama, yaitu (1) Analisis masalah untuk mengetahui apakah permasalahan tersebut dapat diselesaikan dengan metode yang kami gunakan. (2) Merancang algoritma dan mengembangkan program menggunakan *Python Programming Language*. (3) Membandingkan *running time* dari *output* kedua program yang dibangun dari kedua algoritma yang kami gunakan. Hasil penelitian ini menunjukkan bahwa algoritma *binary search* lebih efektif dan efisien daripada algoritma *sequential search* dalam pencarian kartu *royal flush* pada *deck* kartu poker yang baru karena waktu yang dibutuhkan oleh algoritma *binary search* lebih cepat dan lebih stabil dibanding dengan algoritma *sequential search*.

Kata kunci: *brute force, divide and conquer, sequential search, binary search*

BAB I
PENDAHULUAN

Strategi adalah rencana yang cermat mengenai kegiatan untuk mencapai sasaran khusus (KBBI). Algoritma adalah urutan langkah-langkah untuk memecahkan suatu masalah. Strategi algoritmik adalah kumpulan metode atau teknik untuk memecahkan masalah guna mencapai tujuan yang ditentukan, yang dalam hal ini deskripsi metode atau teknik tersebut dinyatakan dalam suatu urutan langkah-langkah penyelesaian. Secara umum, strategi pemecahan masalah dapat dikelompokan sebagai berikut.

Strategi solusi langsung (*direct solution strategies*) seperti algoritma *Brute Force* dan algoritma *Greedy*, strategi berbasis pencarian pada ruang status *(state-space base strategies)* seperti algoritma *Backtracking*, algoritma *Branch* and *Bound*, strategi solusi atas-bawah *(top-down solution strategies)* seperti algoritma *Divide* and *Conquer*, dan strategi solusi bawah-atas *(bottom-up solution strategies)* seperti *Dynamic Programming*.

Dalam studi kasus ini terdapat seorang yang bernama Alex Chindopler, dia tinggal di sekitar Las Vegas dan setiap minggunya dia selalu pergi ke *Casino* untuk mencari pundi-pundi uang untuk keluarganya. Dia selalu bermain permainan poker, karena menurutnya poker adalah permainan yang jelas meskipun beresiko. Pria ini selalu merasa resah ketika mengalami kekalahan dan kehilangan uangnya, maka dari itu dia selalu membawa satu deck kartu poker yang baru untuk melakukan kecurangan, dengan urutan kartu yang selalu sama, setiap dia akan bermain di *Casino* dia akan selalu mengeluarkan kartu *royal flush*, dengan urutan deck kartu poker yang selalu sama, yaitu :

- ["D", 2], ["D", 3], ["D", 4], ["D", 5], ["D", 6], ["D", 7], ["D", 8], ["D", 9], ["D", 10], ["D", J], ["D", Q], ["D", K], ["D", A]
- ["C", 2], ["C", 3], ["C", 4], ["C", 5], ["C", 6], ["C", 7], ["C", 8], ["C", 9], ["C", 10], ["C", J], ["C", Q], ["C", K], ["C", A]
- ["H", 2], ["H", 3], ["H", 4], ["H", 5], ["H", 6], ["H", 7], ["H", 8], ["H", 9], ["H", 10], ["H", J], ["H", Q], ["H", K], ["H", A]
- ["S", 2], ["S", 3], ["S", 4], ["S", 5], ["S", 6], ["S", 7], ["S", 8], ["S", 9], ["S", 10], ["S", J], ["S", Q], ["S", K], ["S", A]

Royal flush merupakan sebuah kombinasi kartu yang memiliki nilai paling besar. Biasanya, *royal flush* tersebut terdiri dari kartu 10, *jack, queen*, *king*, dan *AS*.

Pada tugas besar ini, kami menyelesaikan kasus tersebut menggunakan metode *Brute Force* dan *Divide & Conquer*. Kedua metode ini merupakan metode yang digunakan dalam pencarian kartu *royal flush* dengan akurat. Kemudian kami akan membandingkan kedua metode tersebut untuk menentukan metode manakah yang lebih akurat dan efisien dalam segi pencarian dan waktu.

BAB II
DASAR TEORI

2.1 Studi Kasus

Poker adalah salah satu permainan kartu yang bisa menggunakan kemungkinan algoritma untuk mendapatkan urutan kartu istimewa dalam permainan. Dengan begitu cara bermain poker adalah dengan memahami jumlah kombinasi kartu dari yang tertinggi hingga terendah. Berikut kombinasinya dari yang tertinggi hingga terendah.

- *Royal Flush* (kartu 10, *jack*, *queen*, *king*, dan as, semuanya satu jenis) – paling tinggi nilainya karena paling mengejutkan jika didapatkan. Pemikiran salah yang umum adalah bahwa kombinasi kartu ini paling sulit didapatkan daripada lima kartu serupa lainnya.
- *Straight Flush* (lima kartu dengan angka berurutan, semuanya satu jenis) – tidak boleh memiliki kartu *king* dan kartu dua secara bersamaan (misalnya Q-K-A-2-3).
- *Four of a Kind* (empat kartu dengan angka yang sama dan satu kartu sembarang).
- *Full House* (tiga kartu dengan angka yang sama dan dua kartu dengan angka yang sama) untuk kombinasi kartu yang sama-sama *Full House*, yang lebih kuat ditentukan oleh kartu yang bernilai lebih tinggi dari tiga kartu yang memiliki angka yang sama.
- *Flush* (lima kartu dengan jenis yang sama) – angka berapapun bukan masalah.
- *Straight* (lima kartu dengan angka berurutan, jenis berbeda) – tidak boleh memiliki kartu *king* dan kartu dua secara bersamaan (misalnya J-Q-K-A-2).
- *Three of a Kind* (tiga kartu dengan angka yang sama, dua kartu lain dengan angka berbeda) – jika dua kartu lainnya memiliki angka yang sama, maka akan menjadi *Full House*.
- *Two Pair* (dua pasang kartu dengan angka yang sama ditambah satu kartu dengan angka berbeda).
- *One Pair* (dua kartu dengan angka yang sama, tiga kartu lainnya dengan angka yang berbeda-beda).
- *High Card* (nilai dari kartu poker itu sendiri bisa berupa urutan kartu juga berupa gambar kartu poker)

2.2 Algoritma *Brute Force*

Brute force adalah sebuah pendekatan yang lempang *(straightforward)* untuk memecahkan suatu masalah, biasanya didasarkan pada pernyataan masalah *(problem statement)* dan definisi konsep yang dilibatkan. Algoritma *brute force* memecahkan masalah dengan sangat sederhana, langsung dan dengan cara yang jelas *(obvious way)*.

Cara kerja *brute force* dengan melakukan enumerasi (*list*) setiap solusi yang mungkin dengan cara yang sistematis. *Brute force* akan mengevaluasi setiap kemungkinan solusi satu per satu dan simpan solusi terbaik yang ditemukan sampai sejauh ini. Bila pencarian solusi berakhir, makan akan ditampilkan solusi terbaik.

Algoritma *brute force* sebenarnya bukan algoritma yang efisien, karena membutuhkan jumlah langkah yang besar dalam menyelesaikan dan membutuhkan waktu yang sebanding dengan jumlah langkah penyelesaiannya. Dalam pencarian pola-pola dasar, keteraturan, atau trik-trik khusus, biasanya dapat membantu untuk menemukan algoritma yang lebih cerdas dan lebih mangkus lagi.

Pada program yang kami gunakan ini, kami memakai *sequential search*, salah satu teknik pencarian dengan solusi *brute force* untuk menyelesaikan masalah kombinatorik. Kami akan melakukan enumerasi (*list*) setiap solusi yang mungkin dengan cara yang sistematis. Teknik pencarian data dimana data dicari secara urut dari depan ke belakang atau dari awal sampai akhir berdasarkan data yang dicari.

2.3 Algoritma *Divide and Conquer*

Algoritma *Divide and Conquer* merupakan algoritma yang sangat populer di dunia Ilmu Komputer. *Divide and Conquer* merupakan algoritma yang berprinsip memecah-mecah permasalahan yang terlalu besar menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan. *Divide* adalah membagi masalah menjadi beberapa upa-masalah yang memiliki kemiripan dengan masalah semula namun berukuran lebih kecil, idealnya berukuran hampir sama. *Conquer* adalah memecahkan masing-masing masalah secara rekursif. *Combine* adalah menggabungkan solusi masing-masing upa-masalah sehingga membentuk solusi masalah semula.

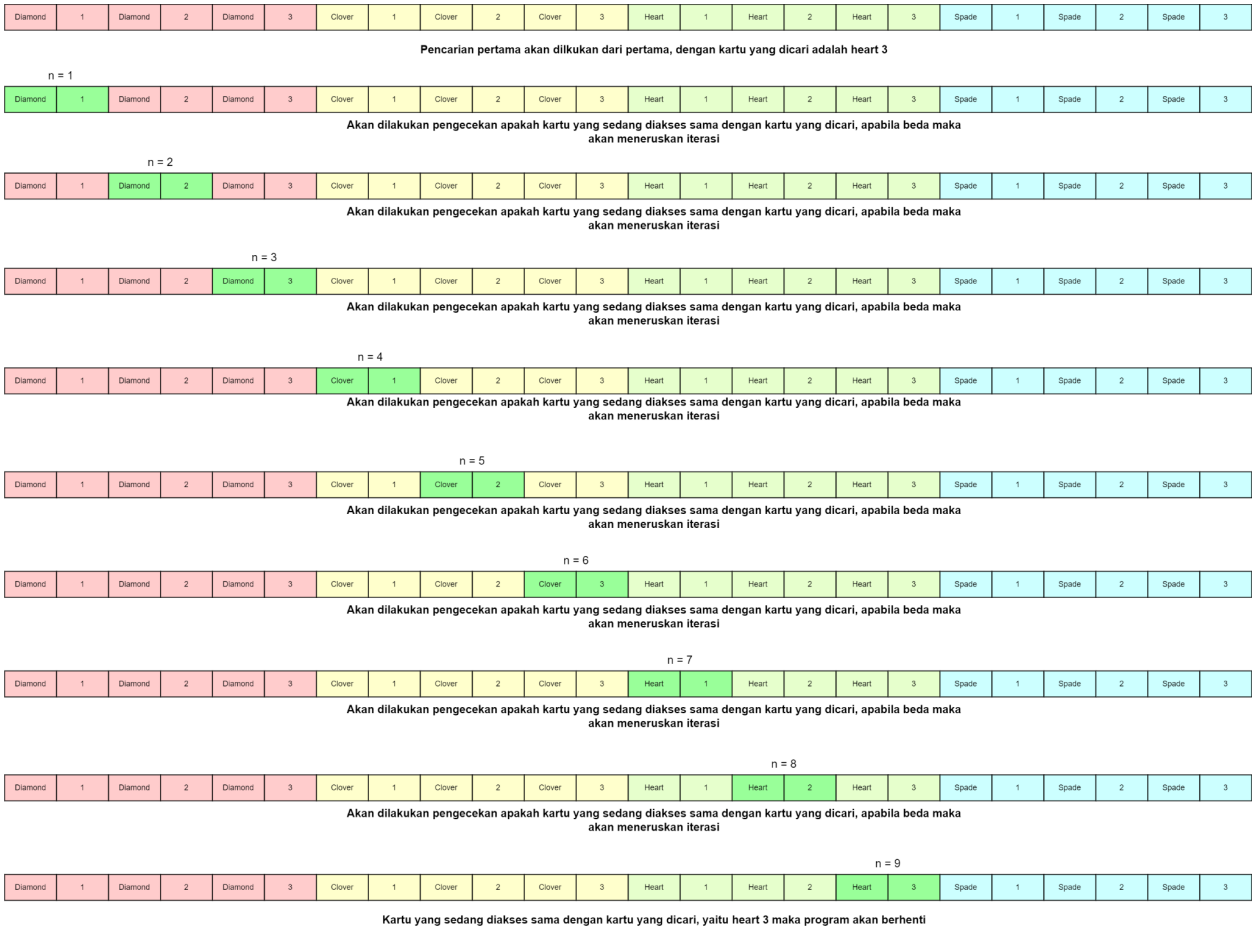
Algoritma *divide and conquer* mempunyai kelebihan yang dapat mengurangi kompleksitas pencarian solusi suatu masalah karena prinsip kerjanya yang membagi-bagi masalah menjadi masalah yang lebih kecil. Kelebihan tersebut banyak menguntungkan dari segi waktu, tenaga, dan sumberdaya. Algoritma *divide and conquer* memecah masalah menjadi submasalah independen yang lebih kecil sehingga solusi submasalah dapat diperoleh secara mudah. Solusi dari sub masalah kemudian digabung menjadi solusi dari seluruh masalah.

Pada program yang kami gunakan ini, kami memakai *binary search*. *Binary search* sendiri adalah algoritma salah satu teknik pencarian untuk data yang terurut. Pencarian dilakukan dengan cara menebak apakah data yang dicari berada ditengah-tengah data, kemudian membandingkan data yang dicari dengan data yang ada ditengah.

BAB III
IMPLEMENTASI

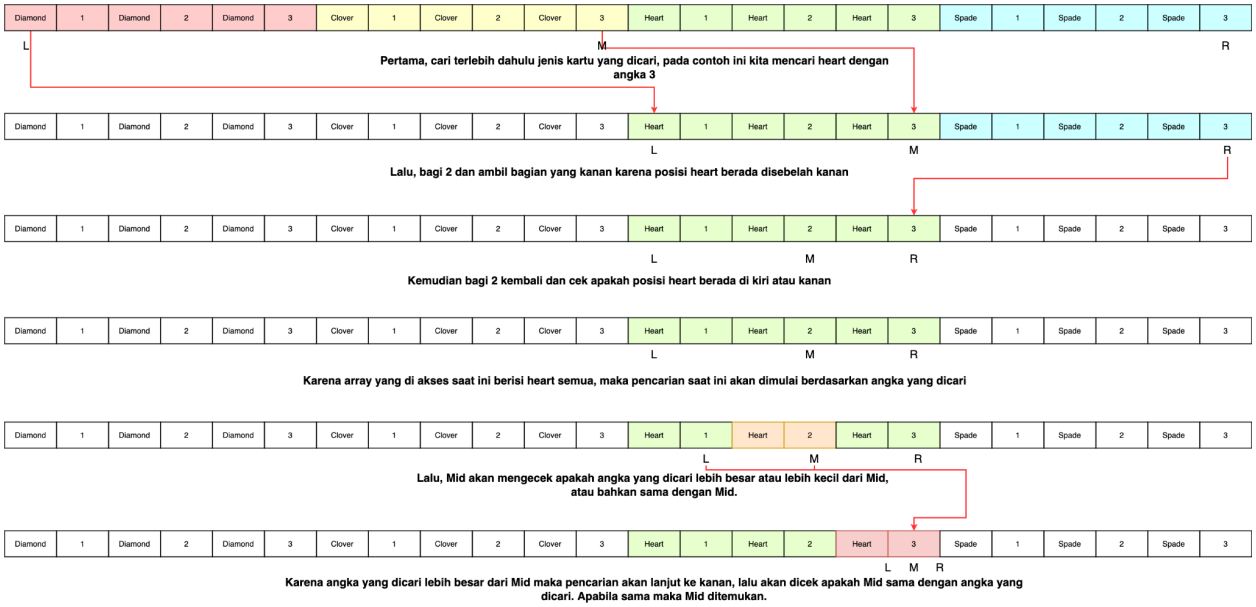
3.1 Cara Kerja *Sequential Search*

Sequential search bekerja dengan cara mengecek data satu persatu secara berurutan hingga algoritma menemukan data yang sesuai dengan data yang dicari. Jika diimplementasikan pada kasus yang kami ambil, asumsi kartu yang dicari adalah kartu *Heart* dengan angka 3, maka program akan mengecek satu persatu sesuai dengan urutan tingkat jenis kartu beserta angkanya hingga program menemukan kartu yang sama dengan kartu dan angka yang dicari yaitu kartu *Heart* dengan angka 3.



3.2 Cara Kerja *Binary Search*

Binary search merupakan algoritma pencarian untuk data terurut yang dilakukan dengan cara berulang kali membagi separuh dari dataset yang dicari hingga memperkecil lokasi pencarian menjadi satu data. *Binary search* akan menebak apakah data yang dicari berada di tengah-tengah *dataset*, kemudian membandingkan data yang dicari dengan data yang ada di tengah. Jika data yang di tengah sama dengan data yang dicari, maka artinya data telah ditemukan. Jika diimplementasikan pada kasus yang kami ambil, program akan mencari jenis kartu yang dicari terlebih dahulu, lalu program mencari bagian tengah yang kemudian akan menjadi kartu yang berada di posisi tengah yang kemudian menjadi titik untuk *dataset* dibagi menjadi dua bagian dan diambil bagian yang terdapat jenis kartu yang dicari di dalamnya. Kemudian program akan melakukan hal tersebut secara berulang-ulang hingga jenis kartu dan angka yang dicari ditemukan.



BAB IV
ANALISIS

4.1 Pseudocode *Sequential Search*

```
Procedure bruteForce(Input check : string, Input tupleCard : array)

{I.S Diterima sebuah array yang memiliki dua value, dan akan dicari
sebuah data yang sesuai dengan ketentuan}

{F.S Menampilkan dilayar data yang dicari di layar}

Kamus

      finalHand : array
      exist : array
      count : integer
      countI : integer
      test : integer
      i : integer

Algoritma

      count ← 0

      countI ← 0

      test ← 0



      for i ← 0 to len(tupleCard) do

            if tupleCard[i][0] = check and (tupleCard[i][1] = 10
                  or tupleCard[i][1] = 11
                  or tupleCard[i][1] = 12
                  or tupleCard[i][1] = 13
                  or tupleCard[i][1] = 14) then
               finalHand.append(tupleCard[i])
               exist.append(1)
            else

                  pass

            {endif}
```

```
            if len(exist) = 5 then
                  for j ← 0 to len(exist) do
                        count ← count + 1
                        if count = 5 then
                              test ← test + 1
                        {endif}
                  {endfor}
            {endif}

            if test = 1 then
                  break
            else
                  pass

            {endif}

      {endfor}


     while countI ≠ len(finalHand) do

           output(finalHand[countI])

           countI ← countI + 1


     {endwhile}

{endprocedure}
```

4.1.1 Perhitungan Kompleksitas Waktu Algoritma *Brute Force*

$$T(n) = \sum_{i=0}^{len(tupleCard)} 1 \in O(len(tupleCard))$$

4.2 Pseudocode *Binary Search*

```
Procedure DnC(Input check : string, Input tupleCard : array)

{I.S Diterima sebuah array yang memiliki dua value, dan akan dicari
sebuah data yang sesuai dengan ketentuan}

{F.S Menampilkan dilayar data yang dicari di layar}

Kamus

      finalHand, current, onHand   : array

      countI, left, mid, right, i  : integer

Algoritma

      countI ← 0

      onHand ← tupleCard

      countI ← 0

      left ← 0

      right ← len(tupleCard)

      mid ← (right + left) / 2


      if check = "H" or check = "S" then

            left ← mid

            if check = "H" then

                  mid ← (right + left) / 2

                  right ← mid

            else if check = "S" then

                  mid ← (right + left) / 2

                  left ← mid

            {endif}

      else if check = "D" or check = "C" then

            right ← mid

            if check = "D" then

                  mid ← (right + left) / 2

                  right ← mid

            else if check = "C" then

                  mid ← (right + left) / 2

                  left ← mid

      {endif}
```

```
i ← left
while i != right do
      current.append(onHand[i])
      i++
{endwhile}

left ← 0
right ← len(current) - 1
mid ← 0

while len(finalhand) != len(royalCardList)do
      for i to len(royalCardList)do
            mid ← (right + left) / 2

            if royalCardList[i] > current[mid][1] then
                  left ← mid + 1
                  mid ← (right + left) / 2
                  while royalCardList[i] > current[mid][1] do
                        left ← mid + 1
                        mid ← (right + left) / 2
                  {endwhile}
            {endif}
```

```
                if royalCardList[i] < current[mid][1] then
                      right ← mid - 1
                      mid ← (right + left) / 2 + 1
                      while royalCardList[i] < current[mid][1] do
                            right ← mid - 1
                            mid ← (right + left) / 2 + 1
                      {endwhile}
                {endif}
                if royalCardList[i] = current[mid][1] then
                      finalHand.append(current[mid])
                      current.remove(current[mid])

                      left ← 0
                      right ← len(current) - 1
                      mid ← 0
                {endif}
          {endfor}
    {endwhile}
    Output(finalHand)
{endprocedure}
```

4.2.1 Perhitungan Kompleksitas Waktu Algoritma *Binary Search*

$$T(n)\begin{cases}0 & ,n = 0\\ 1 + T(\lfloor n/2 \rfloor, & n > 0\end{cases}$$

$T(n) = 1 + T\lfloor n/2 \rfloor$

$T(n) = 1 + (1 + T\lfloor n/4 \rfloor)$

$T(n) = 2 + T\lfloor n/4 \rfloor$

$T(n) = 2 + (1 + T\lfloor n/8 \rfloor)$

$T(n) = 3 + T\lfloor n/8 \rfloor$

Pattern identified

$T(n) = T(n/2^i) + ic$

$T(n/2^i) = T(1)$

$\frac{n}{2^i} = 1$

$n = 2^i$

Mengambil $\log_2$ di 2 sisi

$\log_2 n = \log_2 2^i$

$\log_2 n = i$

$T(n) = i + c \log_2 n$

$T(n) = O(\log_2 n)$

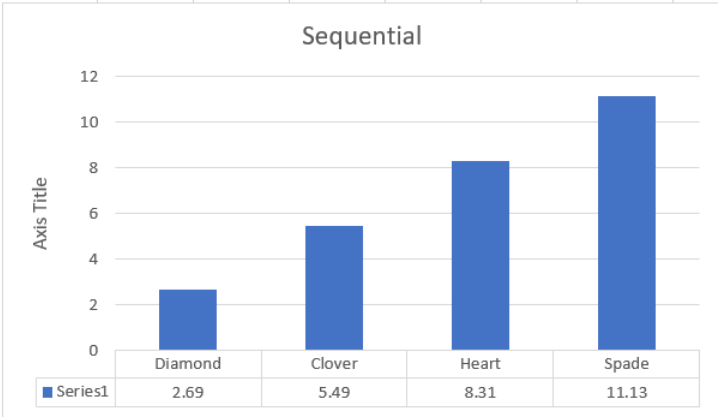
4.3 Perbandingan Waktu *Sequential Search* dan *Binary Search*

4.3.1 Perbandingan *Sequential Search*

Algoritma *sequential search* akan mencari data sesuai kata kunci yang diberikan mulai dari elemen awal pada *array* hingga elemen akhir *array*. Pada percobaan ini kami melakukan uji coba dengan mencari *running time* pada setiap pencarian jenis kartu layaknya *Diamond, Clover, Heart, Spade.* Uji coba dilakukan dua kali untuk memastikan kecepatan waktu pencarian menggunakan *sequential* dalam satuan waktu.

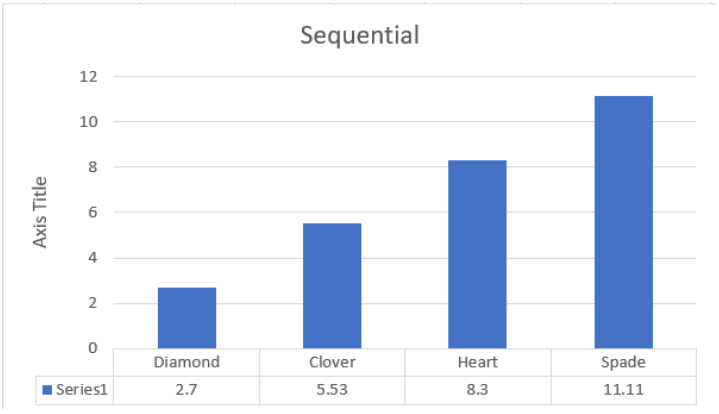
Percobaan 1 (dalam satuan waktu)

- *Diamond* = 2.69
- *Clover* = 5.49
- *Heart* = 8.31
- *Spade* = 11.13



Percobaan 2 (dalam satuan waktu)

- *Diamond* = 2.70
- *Clover* = 5.53
- *Heart* = 8.30
- *Spade* = 11.11

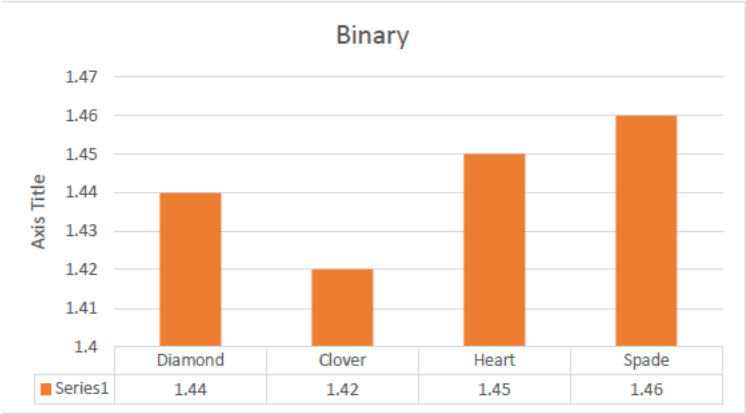


4.3.2 Perbandingan *Binary Search*

Dengan teknik *binary search* akan membuang setengah dari jumlah data. Apabila ditemukan kecocokan data maka program akan menampilkan output, jika tidak pencarian akan terus berlanjut hingga akhir dari pembagian jumlah data tersebut. Pada percobaan ini kami melakukan uji coba dengan mencari *running time* pada setiap pencarian jenis kartu layaknya *Diamond, Clover, Heart, Spade.* Uji coba dilakukan dua kali untuk memastikan kecepatan waktu pencarian menggunakan *binary* dalam satuan waktu.

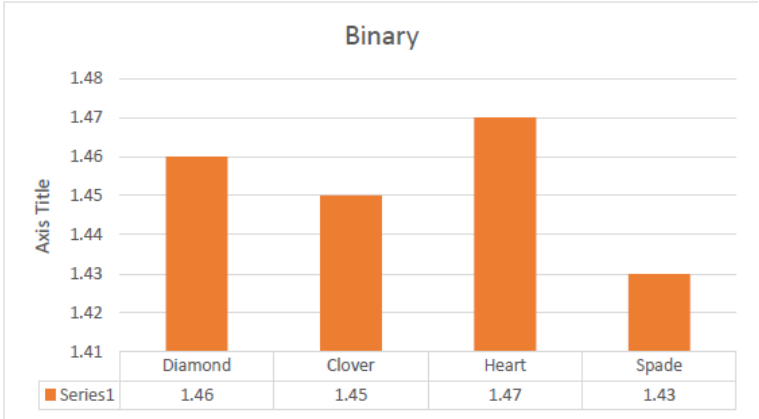
Percobaan 1 (dalam satuan waktu)

- *Diamond* = 1.44
- *Clover* = 1.42
- *Heart* = 1.45
- *Spade* = 1.46



Percobaan 2 (dalam satuan waktu)

- *Diamond* = 1.46
- *Clover* = 1.45
- *Heart* = 1.47
- *Spade* = 1.43

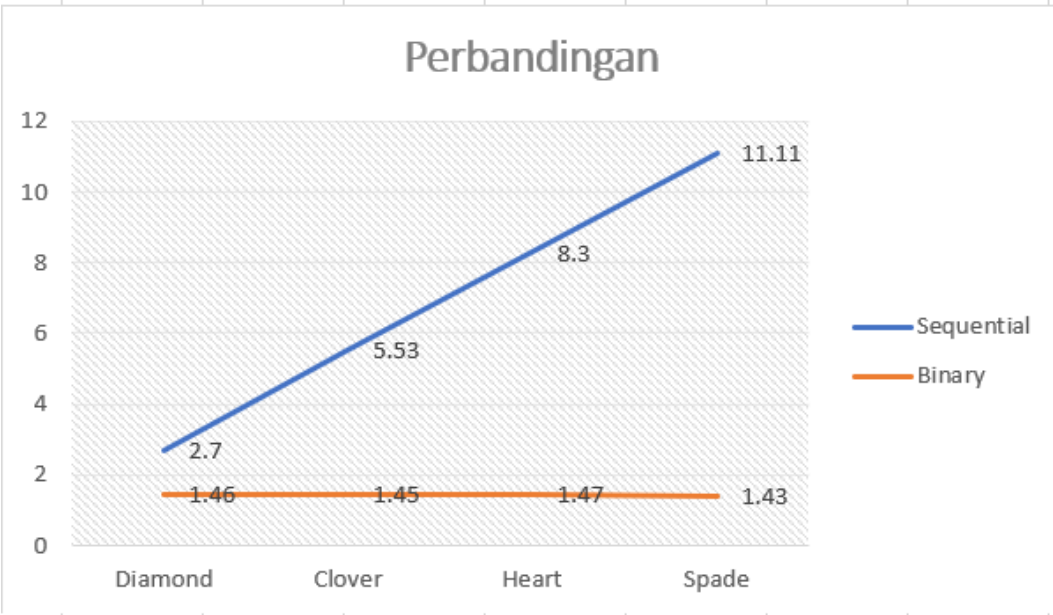


4.3.3 Perbandingan Kedua Algoritma

Pada sub bab ini, kami telah melakukan 4 percobaan, dengan 2 percobaan menggunakan *sequential search* (*brute force*) dan 2 percobaan menggunakan *binary search* (*divide and conquer*). Dari hasil yang didapatkan, kami melakukan perbandingan antara *sequential* dan *binary*. Didapatkan bahwa pencarian *binary* memberikan hasil yang memuaskan, karena waktu yang dibutuhkan dalam pencarian kombinasi *royal flush* lebih cepat dibandingkan *brute force*.

Percobaan 1

Percobaan 2



BAB V
KESIMPULAN

Dari studi kasus diatas, kami melakukan analisis terhadap setiap algoritma yang dipakai *sequential search* dan *binary search* yang menghasilkan kompleksitas waktu dari masing - masing algoritma, yaitu:

- *Sequential Search* = *O(n)*
- *Binary Search* = *O(log n)*

Dari hasil tersebut, kami lalu melakukan perbandingan antara kedua algoritma tersebut. Kami mendapatkan hasil bahwa algoritma *binary search* lebih baik dari *sequential search* dalam pencarian kartu (*O(n) > O(log n)*). Maka dari itu algoritma yang cocok digunakan oleh Alex Chindopler dalam mencari kartu *royal flush* yang diinginkan adalah algoritma *binary search*, karena waktu yang dibutuhkan oleh algoritma *binary search* dalam mencari kartu *royal flush* yang diinginkan lebih cepat dan lebih stabil apabila dibandingkan dengan algoritma *sequential search*.

Daftar Pustaka

Media, A. (2021). *Cara kerja Algoritma Brute Force | Contoh kasus Implementasi Algoritma Brute Force | Creator Media*. [online] Available at: https://creatormedia.my.id/cara-kerja-algoritma-brute-force/ [Accessed 30 May. 2022].

Algoritma Brute Force Desain dan Analisis Algoritma (CS3024). (n.d.). [online] Available at: https://repository.unikom.ac.id/37037/1/BruteForce%28bagian%201%29.pdf [Accessed 31 May. 2022].

Andikafisma's Blog. (2010). *Algoritma Divide and Conquer*. [online] Available at: https://andikafisma.wordpress.com/algoritma-divide-and-conquer/.

Riski (2016). *Apakah Strategi Algoritmik (Algorithm Strategies) Itu?* -. [online] berwirausaha. Available at: https://www.berwirausaha.net/2016/02/apakah-strategi-algoritmik-algorithm-strategies-itu.html/ [Accessed 4 Jun. 2022].

Liputan6.com (2019). *Cara Bermain Poker, Permainan Kartu Paling Populer di Dunia Tanpa Berjudi*. [online] liputan6.com. Available at: https://www.liputan6.com/citizen6/read/3919619/cara-bermain-poker-permainan-kartu-paling-populer-di-dunia-tanpa-berjudi [Accessed 5 Jun. 2022].

Lampiran

Link Source Code: https://github.com/mobs3288/tugas-besar-strategi-algoritma-CII2K3

- *Card initialization*

```
# Kami mendefinisikan kartu jack, queen, king, ace sebagai berikut
'''
Jack    = 11
Queen   = 12
King    = 13
Ace     = 14
'''

# List dari masing masing kartu
diamondTupleCard = ( ["D", 2], ["D", 3], ["D", 4], ["D", 5], ["D", 6], ["D", 7], ["D", 8], ["D", 9], ["D", 10], ["D", 11], ["D", 12], ["D", 13], ["D
cloverTupleCard  = ( ["C", 2], ["C", 3], ["C", 4], ["C", 5], ["C", 6], ["C", 7], ["C", 8], ["C", 9], ["C", 10], ["C", 11], ["C", 12], ["C", 13], ["C
heartTupleCard   = ( ["H", 2], ["H", 3], ["H", 4], ["H", 5], ["H", 6], ["H", 7], ["H", 8], ["H", 9], ["H", 10], ["H", 11], ["H", 12], ["H", 13], ["H
spadeTupleCard   = ( ["S", 2], ["S", 3], ["S", 4], ["S", 5], ["S", 6], ["S", 7], ["S", 8], ["S", 9], ["S", 10], ["S", 11], ["S", 12], ["S", 13], ["S

# Kombinasi kartu yang dianggap sebagai royal flush
royalCardList = [10, 11, 12, 13, 14]

# Deck kartu yang telah disatukan
tupleCard = diamondTupleCard + cloverTupleCard + heartTupleCard + spadeTupleCard
```

- *Binary Search*

```
# Mencari kartu royal flush dimana jenis kartu telah ditentukan dengan metode divide and conquer dengan algoritma binary search
def binarySearch(check, tupleCard):
    finalHand = []
    onHand = list(tupleCard)
    current = []
    countI = 0

    left = 0
    right = len(tupleCard)
    mid = int((right + left) / 2)

    start = time.time()
    time.sleep(1)

    # Mencari lambang kartu yang sesuai dengan metode divide and conquer
    if check == "H" or check == "S":
        time.sleep(0.07)
        left = mid

        if check == "H":
            mid = int((right + left) / 2)
            right = mid

        elif check == "S":
            mid = int((right + left) / 2)
            left = mid

    elif check == "D" or check == "C":
        right = mid

        if check == "D":
            mid = int((right + left) / 2)
            right = mid
```

```
    elif check == "C":
        mid = int((right + left) / 2)
        left = mid

i = left
while i != right:
    current.append(onHand[i])
    i += 1

left = 0
right = len(current) - 1
mid = 0

# Mencari pasangan kartu royal flush dengan metode divide and conquer
while len(finalHand) != len(royalCardList):
    time.sleep(0.15)
    for i in range(len(royalCardList)):
        time.sleep(0.15)
        mid = (right + left) // 2

        # Jika royalCardList[i] > current[mid][1] maka bagian kiri akan diabaikan
        if royalCardList[i] > current[mid][1]:
            left = mid + 1
            mid = (right + left) // 2
            while royalCardList[i] > current[mid][1]:
                time.sleep(0.15)
                left = mid + 1
                mid = (right + left) // 2
```

```
            # Jika royalCardList[i] < current[mid][1] maka bagian kanan akan diabaikan
            if royalCardList[i] < current[mid][1]:
                right = mid - 1
                mid = (right + left) // 2 + 1
                while royalCardList[i] < current[mid][1]:
                    time.sleep(0.15)
                    left = mid - 1
                    mid = (right + left) // 2 + 1

            # # Jika royalCardList[i] = current[mid][1] maka akan dimasukkan ke finalHand
            if current[mid][1] == royalCardList[i]:
                finalHand.append(current[mid])
                current.remove(current[mid])

                left = 0
                right = len(current) - 1
                mid = 0

end = time.time()

checkTime = ((end - start) - 1 - 0.1)

print("\nFinal hand : ")

# Menampilkan seluruh kartu yang ada di finalHand
while countI != len(finalHand):
    time.sleep(0.1)
    print(finalHand[countI])
    countI += 1

print(f"\nRuntime of Divide and Conquer is {checkTime}")
```

- *Sequential Search*

```
# Mencari kartu royal flush dimana jenis kartu telah ditentukan dengan metode bruteforce dengan algoritma sequential search
def bruteForce(check, tupleCard):
    finalHand = []
    exist = []
    count = 0
    countI = 0
    test = 0

    start = time.time()
    time.sleep(1)
    # Looping sebanyak panjang deck kartu
    for i in range(len(tupleCard)):
        time.sleep(0.2)

        print(" __________")
        print(f"|{tupleCard[i][1]}")
        print(f"|{tupleCard[i][0]}         |")
        print("|          |")
        print("|          |")
        print(f"|        {tupleCard[i][1]}")
        print(f"|        {tupleCard[i][0]} |")
        print(" ----------")

        # Apabila ada kartu yang dicari maka akan ditambahkan ke finalHand
        if tupleCard[i][0] == check and (tupleCard[i][1] == 10 or tupleCard[i][1] == 11 or tupleCard[i][1] == 12 or tupleCard[i][1] == 13 or tupleCa
            finalHand.append(tupleCard[i])
            exist.append(1)
        else:
            pass
```

```
        if len(exist) == 5:
            for j in range(len(exist)):
                if exist[j] == 1:
                    count += 1
                    if count == 5:
                        test += 1
        if test == 1:
            break
        else:
            pass

    end = time.time()

    checkTime = ((end - start) - 1 - 0.1)

    print("\nFinal hand : ")

    # Menampilkan seluruh kartu yang ada di finalHand
    while countI != len(finalHand):
        time.sleep(0.1)
        print(finalHand[countI])
        countI += 1

    print(f"\nRuntime of Brute Force is {checkTime}")
```

- *Check Input*

```
# Fungsi untuk mengecek input user dan akan mengubahnya menjadi satu huruf
def checkInput(x):
    if inputCard == "diamond":
        x = "D"
    elif inputCard == "clover":
        x = "C"
    elif inputCard == "heart":
        x = "H"
    elif inputCard == "spade":
        x = "S"
    return x
```

- *Main Function*

```
# Fungsi main
if __name__ == "__main__":
    menu = ""
    inputJenis = ""

    while menu != "n":
        print("_____")
        print("                MUBES")
        print("--------------------------------------")

        print("Diamond | Clover | Heart | Spade\n")

        inputCard = input("Masukkan Jenis Kartu : ").lower()

        # Looping dibawah untuk mengecek apakah input yang dimasukkan oleh user itu valid atau tidak, jika tidak maka akan minta input ulang
        while inputCard != "diamond" and inputCard != "clover" and inputCard != "heart" and inputCard != "spade":
            clear()
            print("Salah Gan, Coba input lagi!")
            print("_____")
            print("                MUBES")
            print("--------------------------------------")

            print("Diamond | Clover | Heart | Spade\n")

            inputCard = input("Masukkan Jenis Kartu : ").lower()

        check = checkInput(inputCard)

        print("_____")
        print("Pilih Jenis Search")
        print("1 : Bruteforce Search")
        print("2 : Binary Search")
        print("--------------------------------------")
```

```
        # Kondisi untuk memanggil fungsi terkait
        if inputJenis == "1":
            bruteForce(check, tupleCard)
        elif inputJenis == "2":
            binarySearch(check, tupleCard)

        # Input untuk meminta user akankah user mengulang sistem kembali atau tidak.
        menu = input("Mulai dari awal lagi? (Y/N) : ").lower()

        clear()
```