LAPORAN STUDI KASUS ANALISIS SORTING
ANALISIS KOMPLEKSITAS ALGORITMA

Disusun oleh:

Dewa Made Wijaya (1301204106)

M Aziz Al Adro Jalil (1301202611)

Program Studi Informatika

Fakultas Informatika

Telkom University 2021

BAB I

Pendahuluan

1.1 latar belakang

Dalam suatu array, kita kerap kali ingin mencari nilai yang tertinggi ataupun nilai terendah. Hal ini tentu saja menjadi salah satu hambatan kita untuk menentukannya karena keterbatasan analisa terhadap suatu array. Ini akan menjadi masalah ketika jumlah array yang di analisis sangat banyak. Suatu hal yang sangat memakan waktu jika kita akan menghitung satu persatu data nya. Tidak hanya itu, mungkin saja kita dapat mengalami kekeliruan saat melakukan analisis datanya. Selain mencari data yang tertinggi dan terendah dari suatu array kita juga ingin melakukan pengurutan terhadap suatu data dalam program. Tentu saja kembali lagi kepada waktu, maka dari permasalahan tersebut, disini kita akan menekankan terkait kompleksitas waktu.

Untuk mengurutkan suatu array, ada banyak sekali algoritma-algoritma yang tersedia dan siap untuk digunakan. Beberapa dari algoritma tersebut yang akan kami analisis pada laporan ini adalah algoritma selection sort dan juga quick sort. Nah, untuk mengetahui kompleksitas waktu dari kedua algoritma tersebut kami bisa menggunakan salah satu materi yang telah di berikan dan di ajarkan pada mata kuliah analisis kompleksitas algoritma yaitu time complexity. Untuk menggunakan time complexity kami membutuhkan algoritma baik recursive maupun non recursive.

Selain itu, tujuan lain kami membuat laporan ini adalah untuk memenuhi salah satu tugas besar dari mata kuliah analisis kompleksitas algoritma yang lebih tepatnya dalam materi time complexity. Karena kami membutuhkan praktik dan implementasi langsung bagaimana time complexity ini dapat di lakukan dalam kehidupan sehari-hari dan pada real case sehingga memudahkan kami untuk memilih algoritma mana yang sekiranya dapat digunakan guna menghemat waktu eksekusi program dengan memperhatikan order of growth dari hasil analisis program atau algoritma yang telah kami buat atau sediakan.

BAB II
Kajian Pustaka

2.1 Algoritma Selection Sort

Selection sort merupakan teknik sorting yang paling sederhana, hal pertama yang akan dilakukan algoritma selection sort adalah menemukan elemen terkecil dalam array kita dan menukarnya (swap) dengan elemen yang ada di posisi pertama, kemudian algoritma ini akan mengulangi hal yang sama lagi yaitu mencari elemen terkecil yang ada di dalam array dan kemudian menukarnya (swap) dengan elemen yang ada di posisi kedua (mengingat elemen di posisi pertama sudah berhasil kita sorting). Proses ini akan terus berlanjut sampai semua elemen yang ada di dalam array telah berhasil kita sorting.

2.1.1 Ilustrasi Algoritma Selection sort

Anggap-lah kita mempunyai 8 elemen array yang masih tidak berurutan seperti ini

Selection sort akan mencari elemen terkecil dari array yang kita punya, dimana disini elemen terkecil adalah 10

Setelah berhasil mendapatkan elemen terkecil, maka selection sort ini akan menukar elemen tersebut ke posisi pertama (10 dan 14 ditukar)

Proses pengurutan pertama berhasil dilakukan, maka selection sortini akan mengulangi lagi hal yang sama, yaitu mencari elemen terkecil yang ada di dalam array kita

Ternyata 14 merupakan elemen terkecil yang ada di dalam array, maka selection sort akan menukar posisi elemen 14 ke posisi kedua (mengingat kita sudah berhasil meletakan elemen terkecil di posisi pertama)

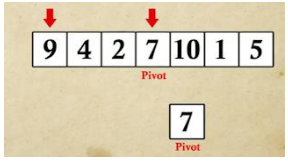
Elemen 14 sudah berhasil kita tukar ke posisi kedua, maka dengan itu proses pengurutan kedua berhasil kita lakukan dan selection sort akan melanjutkannya ke proses pengurutan selanjutnya

2.2 Algoritma Quick Sort

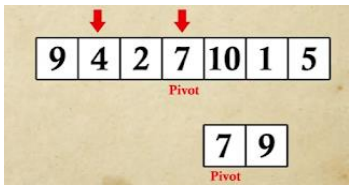
Quick Sort merupakan suatu algoritma pengurutan data yang menggunakan teknik pemecahan data menjadi partisi-partisi, sehingga metode ini disebut juga dengan nama partition exchange sort. Untuk memulai iterasi pengurutan, pertama-tama sebuah elemen dipilih dari data, kemudian elemen-elemen data akan diurutkan diatur sedemikian rupa.

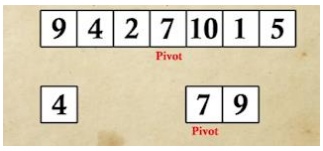
Algoritma ini mengambil salah satu elemen secara acak (biasanya dari tengah) yang disebut dengan pivot lalu menyimpan semua elemen yang lebih kecil di sebelah kiri pivot dan semua elemen yang lebih besar di sebelah kanan pivot. Hal ini dilakukan secara recursive terhadap elemen di sebelah kiri dan kanannya sampai semua elemen sudah ter urut.

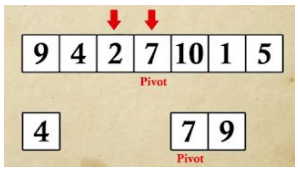
2.2.1 Ilustrasi Algoritma Quick Sort

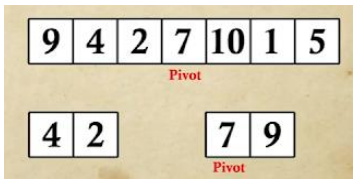


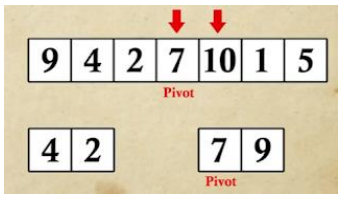

9 4 2 7 10 1 5
Pivot
7 9
Pivot

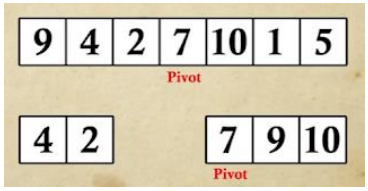

9 4 2 7 10 1 5
Pivot
7 9
Pivot

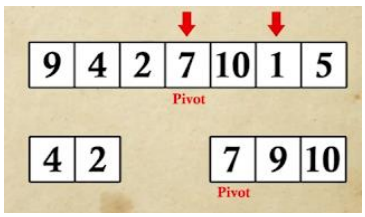

9 4 2 7 10 1 5
Pivot
4
7 9
Pivot

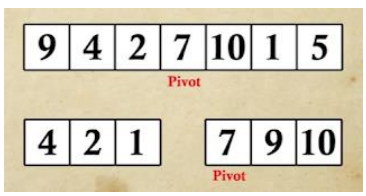

9 4 2 7 10 1 5
Pivot
4
7 9
Pivot

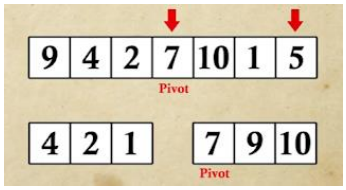

9 4 2 7 10 1 5
Pivot
4 2
7 9
Pivot

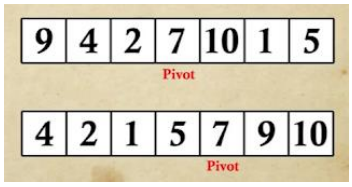

9 4 2 7 10 1 5
Pivot
4 2
7 9
Pivot


9 4 2 7 10 1 5
Pivot
4 2
7 9 10
Pivot

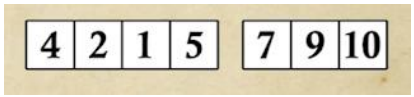

9 4 2 7 10 1 5
Pivot
4 2
7 9 10
Pivot


9 4 2 7 10 1 5
Pivot
4 2 1
7 9 10
Pivot

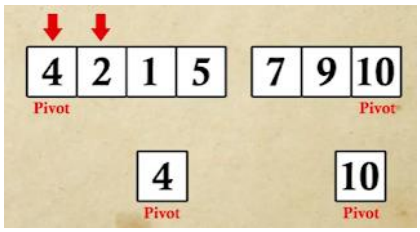

9 4 2 7 10 1 5
Pivot
4 2 1
7 9 10
Pivot

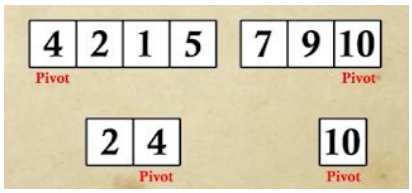

9 4 2 7 10 1 5
Pivot
4 2 1 5 7 9 10
Pivot

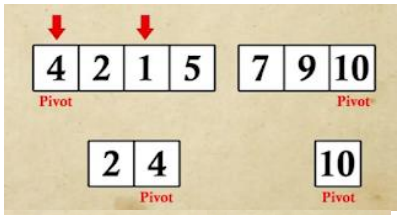

4 2 1 5 7 9 10
Partisi 1 [i]
Partisi 2 [j]

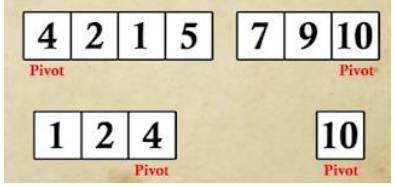

4 2 1 5
7 9 10

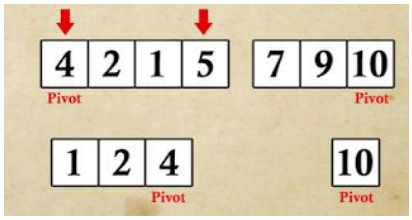

4 2 1 5
Pivot
7 9 10
Pivot

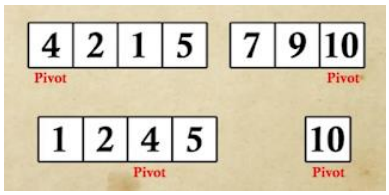

4 2 1 5
Pivot
7 9 10
Pivot
4
Pivot
10
Pivot


4 2 1 5
Pivot
7 9 10
Pivot
2 4
Pivot
10
Pivot


4 2 1 5
Pivot
7 9 10
Pivot
2 4
Pivot
10
Pivot


4 2 1 5
Pivot
7 9 10
Pivot
1 2 4
Pivot
10
Pivot


4 2 1 5
Pivot
7 9 10
Pivot
1 2 4
Pivot
10
Pivot


4 2 1 5
Pivot
7 9 10
Pivot
1 2 4 5
Pivot
10
Pivot


1 2 4 5 7 9 10

BAB III

Data dan Pengujian

3.1 selection sort

3.1.1 Algoritma selection sort

Berikut adalah lampiran algoritma yang kami gunakan dalam code untuk membuat program selection sort

```
function Selection_sort()
{
   c_delay=0;

   for(var i=0;i<array_size-1;i++)
   {
      div_update(divs1[i],div_sizes1[i],"red");//Color update

      index_min=i;

      for(var j=i+1;j<array_size;j++)
      {
         div_update(divs1[j],div_sizes1[j],"blue");//Color update

         if(div_sizes1[j]<div_sizes1[index_min])
         {
            if(index_min!=i)
            {
               div_update(divs1[index_min],div_sizes1[index_min],"cyan");//Color update
            }
            index_min=j;
            div_update(divs1[index_min],div_sizes1[index_min],"red");//Color update
         }
         else
         {
            div_update(divs1[j],div_sizes1[j],"cyan");//Color update
         }
      }

      if(index_min!=i)
      {
         var temp=div_sizes1[index_min];
         div_sizes1[index_min]=div_sizes1[i];
         div_sizes1[i]=temp;

         div_update(divs1[index_min],div_sizes1[index_min],"red");//Height update
         div_update(divs1[i],div_sizes1[i],"red");//Height update
         div_update(divs1[index_min],div_sizes1[index_min],"cyan");//Color update
```

```
        }
        div_update(divs1[i],div_sizes1[i],"green");//Color update
    }
    div_update(divs1[i],div_sizes1[i],"green");//Color update

    enable_buttons();

}
```

Tetapi pada dasarnya, algoritma selection sort yang lebih umum adalah seperti berikut

```
func selectionSort(array []int) {
    for i := 0; i < len(array)-1; i++ {
      min := i
      for j := i + 1; j <= len(array)-1; j++ {
       if array[j] < array[min] {
        min = j
       }
      }
      swap(array, i, min)
    }
}
```

Karena algoritma pada selection sort yang kami gunakan non recursive maka kita akan dapat menghitung C(n)nya menggunakan metode analisis matematis pada algoritma non recursive, seperti berikut:

Input size: n (jumlah atau array yang akan di urutkan)

Basic Operation: Basic operation nya adalah **perbandingan** dua array

Atau lebih tepatnya **div_sizes1[j]<div_sizes1[index_min]**

Banyak waktu basic operation di eksekusi dalam loop terdalam: 1 kali

Maka akan kita peroleh hasil perhitungan seperti di bawah ini:

$$
\begin{aligned}
C(N) &= \sum_{i=0}^{n-1} \sum_{i+1}^{n-1} 1 \\
&= \sum_{i=0}^{n-1} (n-1) - (i+1) + 1 \\
&= \sum_{i=0}^{n-1} n - 1 - i \\
&= \sum_{i=0}^{n-1} (n-1) - \sum_{i=0}^{n-1} i \\
&= (n-1) - \frac{(n-1)n}{2} \\
&= \frac{3n - n^2}{2} - 1 \ \in O(n^2)
\end{aligned}
$$

3.2 Quick Sort

3.2.1 Algoritma Quick Sort

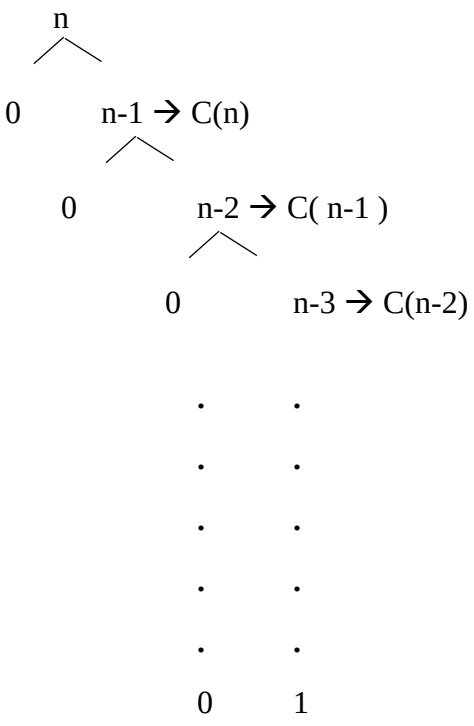
Berikut adalah lampiran algoritma yang kami gunakan dalam code untuk membuat program quick sort

```
function quick_sort (start, end )
{
   if( start < end )
   {
      //stores the position of pivot element
      var piv_pos = quick_partition (start, end ) ;
      quick_sort (start, piv_pos -1);//sorts the left side of pivot.
      quick_sort (piv_pos +1, end) ;//sorts the right side of pivot.
   }
}
```

Karena algoritma pada quick sort ini kami menggunakan algoritma recursive, maka dapat kita hitung C(n) menggunakan metode analisis matematis pada algoritma recursive seperti berikut:

Worst Case :

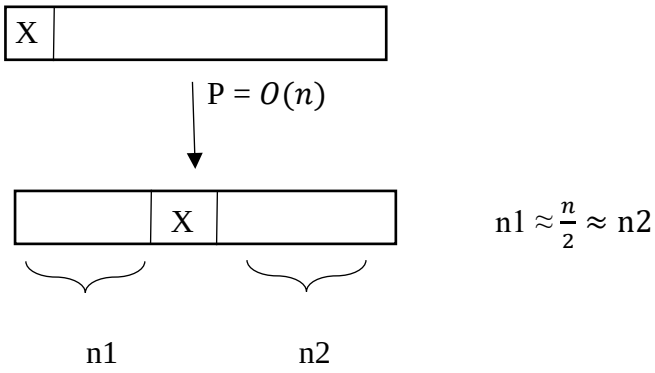
T(n) = T (0) + T(n-1) + C(n)



Mengingat n + (n-1) + (n -2) + . . . + 1

$= = \frac{n\,(n+1)}{2} = \frac{n^2+n}{2} \; \epsilon \; O(n^2)$

Best Case :



n1 $\approx \frac{n}{2} \approx$ n2

T(n) = T($^n/_2$) + $T(^n/_2)$ + $C(n)$

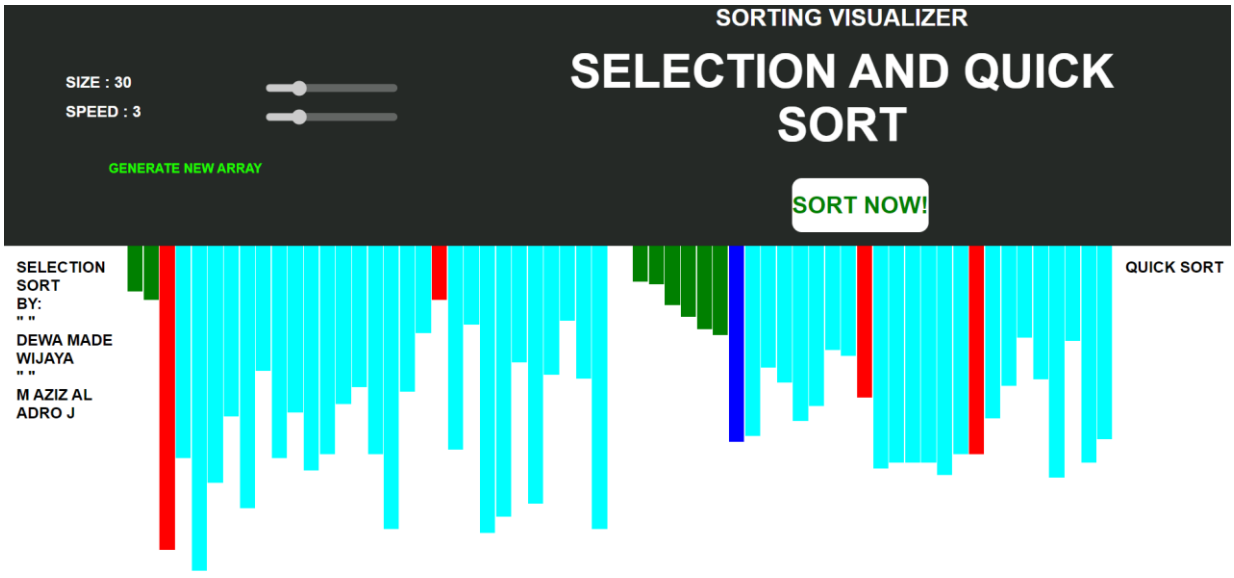
T(n) = 2T($^n/_2$) + $C(n)$ ⟶ Merge Sort

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

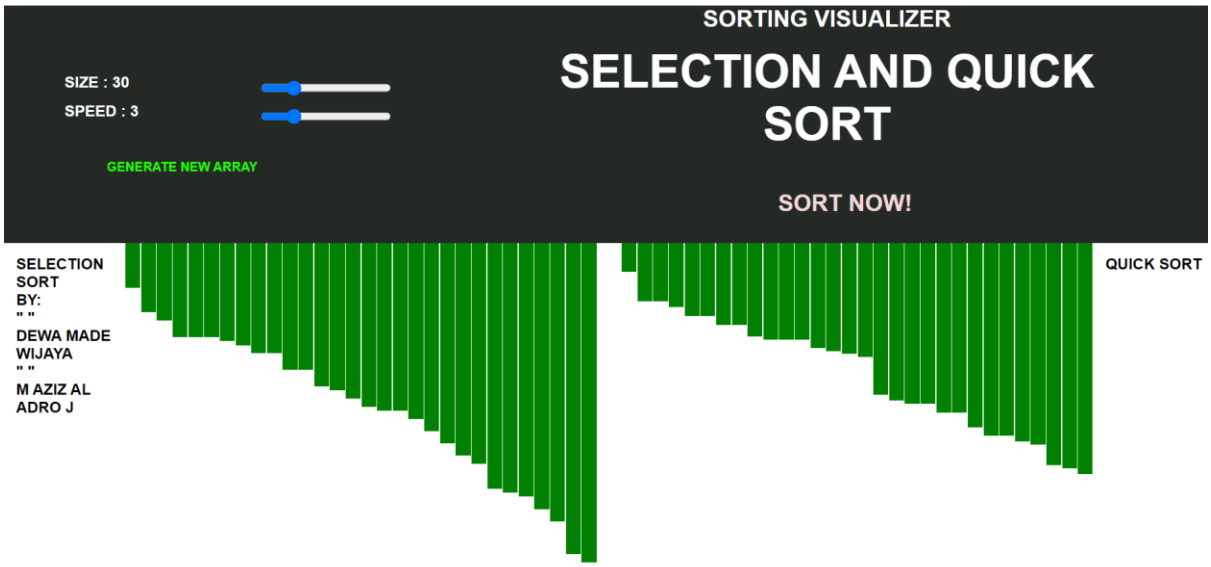
3.2 Visualisasi Perbedaan running time a Selection Sort dan Quick Sort dengan menggunakan aplikasi yang telah dibuat

Aplikasi yang kami buat adalah aplikasi berbasis Web dengan menggunakan bahasa pemrograman HTML, CSS dan JavaScript. Aplikasi yang telah kami buat menyediakan berbagai size array yang dapat dipilih sesuai dengan keperluan. Tidak hanya itu, kami juga menambahkan fitur speed pada aplikasi ini yang dimana pengguna dapat menentukan kecepatan sesuai dengan size array. Berikut adalah gambaran aplikasi yang belum di run :

Dan ketika tombol sort now! Di tekan maka aplikasi akan melakukan pengurutan data seusai dengan kecepatan dan size array yang telah di tentukan seperti gambar di bawah:



Nah, gambar dibawah artinya bahwa aplikasi sudah selesai melakukan proses pengurutan atau sorting.



3.3 Tabel dan Grafik yang menggambarkan perubahan running time antara algoritma selection sort dengan quick sort dengan parameter n (input size)

Dengan menggunakan aplikasi visualisasi running time, terdapat speed melakukan looping yang dapat diatur dari skala 1 sampai 5 dan mengatur input size dari skala 10 – 100. dengan memberikan batas maksimal dari suatu input an 100 agar terlihat signifikan perbedaan antara algoritma dan juga dapat melihat lebih jelas pergerakan dari setiap sorting yang dilakukan oleh program. Dengan demikian, untuk pembuatan tabel kami menggunakan speed 4 dan beberapa input size yang mana akan menggambarkan perbedaan antara algoritma Selection Sort dengan Quick Sort. Pada perhitungan running time kami menggunakan bantuan berupa timer millisecond.

Tabel dan Grafik Running Time antara Selection sort dan Quick Sort

Input Size (N)	Selection Sort	Quick Sort
10	179 ms	179 ms
20	828 ms	709 ms
30	2081 ms	1530 ms
40	3710 ms	1735 ms
50	5589 ms	2243 ms
100	21403 ms	5719 ms

BAB IV
Kesimpulan

Dari hasil yang telah kami lakukan maka dapat kami simpulkan bahwa, running time antara algoritma selection sort dan quick sort memiliki kecepatan dalam melakukan proses pengurutan atau sorting yang berbeda. Dimana pada algoritma selection sort memiliki kecepatan yang lebih lama dibandingkan dengan quick sort. Argumen ini diperkuat dengan hasil perhitungan dari kompleksitas waktu yang telah kami hitung dimana kompleksitas waktu yang dimiliki oleh selection sort lebih besar yaitu dengan order of growth O€ (n^2) sedangkan pada algoritma quick sort memiliki order of growth O€ (n lg n) pada keadaan best case. Maka dari itu kesimpulan sederhana dapat ditarik disini, yaitu algoritma quick sort lebih cepat di bandingkan dengan algoritma selection sort walaupun pada keadaan awal kedua algoritma tersebut mempunyai kecepatan yang hampir sama, tetapi ketika input size bertambah banyak dan dari sanalah mulai terlihat algoritma yang memiliki grafik lebih cepat dan lebih lambat.

Link GitHub :

https://github.com/mazizaladrojalil/Sort-Visualizer

Link Youtube :

https://youtu.be/CmKTQUUlEd4