LAPORAN PRAKTIKUM

STRUKTUR DATA DAN ALGORITMA

ANALISIS ASIMTOTIK

DISUSUN OLEH

Muhammad Alwiza Ansyar M0520051

PROGRAM STUDI INFORMATIKA

FAKULTAS MATEMATIKA DAN ILMU PENGETAHUAN ALAM
UNIVERSITAS SEBELAS MARET

2021

1. Algoritma-1.cpp

a) Source code:

algoritma-1.cpp | algoritma-2.cpp | algoritma-3.cpp

```
1   #include<iostream>
2   #include<time.h>
3   #include<chrono>
4   #include <array>
5   using namespace std;
6   using namespace chrono;
7   void swap(int * xp, int * yp){
8       int temp = *xp;
9       *xp = *yp;
10      *yp = temp;
11  }
12  void STAY(int arr[], int n){
13      int i, j;
14      for(i=0; i<n-1; i++){
15          for(j=0; j<n-i-1; j++){
16              if(arr[j]>arr[j+1]) swap(&arr[j], &arr[j+1]);
17          }
18      }
19  }
```

b) Analisa Big O

Terdapat dua fungsi yaitu fungsi **swap** dan fungsi **STAY**. Fungsi **swap** melakukan pertukaran value antara dua variable, sedangkan fungsi **STAY** melakukan pengurutan elemen array dari kecil ke besar dengan cara memanggil **swap** apabila ditemukan elemen yang tidak urut (dengan cara membandingkan nilai dari elemen)

Fungsi **swap**:

- Memiliki tige baris (8-10) yang masing-masing merupakan konstan sehingga memiliki O(1)

Fungsi **STAY**:

- Baris 13 merupakan konstan sehingga memiliki O(1)
- Baris 14 merupakan loop menggunakan for statement dengan iterasi i++ sehingga memiliki O(n)
- Baris 15 merupakan loop menggunakan for statement yang juga merupakan inner loop dari loop sebelumnya, menggunakan iterasi j++ sehingga memiliki O(n)
- Baris 16 berisi pertukaran yang eksekusinya tergantung dari isi array

c) Analisa kasus

Isi dari array akan mempengaruhi kompleksitas waktu dari program. Jika ditemukan elemen [j] lebih besar dari elemen [j+1], maka akan dipanggil fungsi **swap**. Namun, fungsi **swap** hanya memiliki O(1) sehingga perbedaan antar kasus tidak signifikan

- Bestcase → Array dengan elemen urut dari kecil ke besar, tidak dilakukan **swap** sama sekali → $O(n^2)$
- Worstcase → Array dengan elemen urut dari besar ke kecil, dilakukan **swap** sebanyak n-1 kali → $O(n^2)$
- Averagecase → Array dengan elemen yang acak, **swap** mengikuti elemen → $O(n^2)$

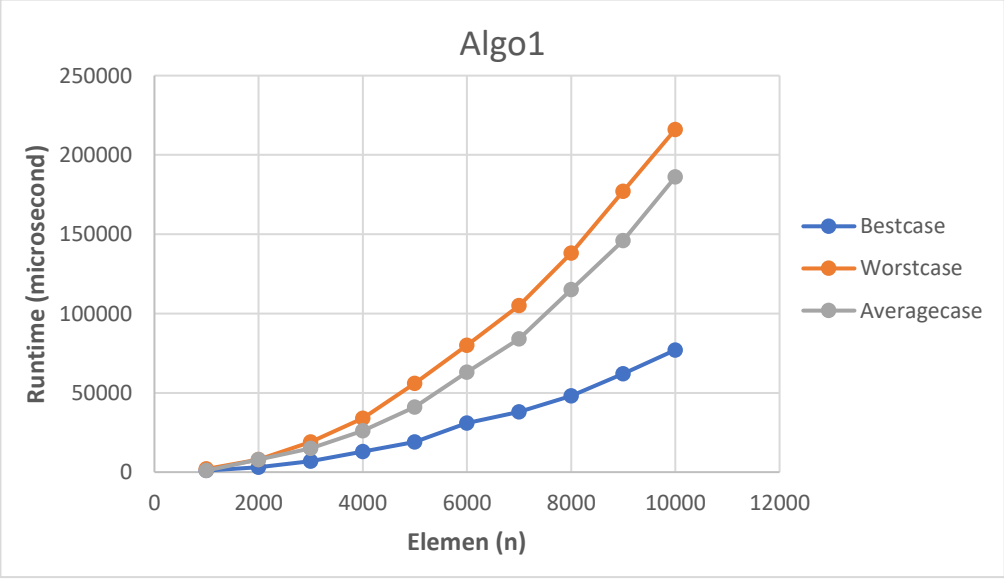
d) Testing

Pada test ini digunakan n dengan initial value 999, batas n<=10000, dan iterasi n=n+1000

```cpp
int main(void){

    cout << "\tBestcase\tWorstcase\tAveragecase" << endl;
    for( int n=999; n<=10000; n+=1000 ){
        int* best = new int[n];
        int* worst = new int[n];
        int* avg = new int[n];

        for( int i=0; i<n; i++ ){
            best[i] = i;
            worst[i] = n-i;
            avg[i] = rand();
        }

        auto best_t0 = high_resolution_clock::now();
        STAY(best, n);
        auto best_t1 = high_resolution_clock::now();
        auto best_dt = best_t1-best_t0;
        long long best_dtms = duration_cast <microseconds> (best_dt).count();

        auto worst_t0 = high_resolution_clock::now();
        STAY(worst, n);
        auto worst_t1 = high_resolution_clock::now();
        auto worst_dt = worst_t1-worst_t0;
        long long worst_dtms = duration_cast <microseconds> (worst_dt).count();

        auto avg_t0 = high_resolution_clock::now();
        STAY(avg, n);
        auto avg_t1 = high_resolution_clock::now();
        auto avg_dt = avg_t1-avg_t0;
        long long avg_dtms = duration_cast <microseconds> (avg_dt).count();

        cout << "n=" << n << "\t" << best_dtms << "\t\t" << worst_dtms << "\t\t" << avg_dtms << endl;
    }
}
```

```
D:\Kuliah\Semester 2\Struktur Data & Algoritma\prak\algoritma-1.exe
        Bestcase        Worstcase       Averagecase
n=999   1010            2010            1000
n=1999  3001            8001            8001
n=2999  7000            19004           15003
n=3999  13003           34007           26006
n=4999  19021           56002           41019
n=5999  31006           80017           63014
n=6999  38009           105022          84019
n=7999  48011           138048          115008
n=8999  62027           177026          146034
n=9999  77010           216048          186061

--------------------------------
Process exited after 1.974 seconds with return value 0
Press any key to continue . . .
```

e) Grafik



2. Algoritma-2.cpp

a) Source code:

algoritma-1.cpp | algoritma-2.cpp | algoritma-3.cpp

```cpp
#include<iostream>
#include<time.h>
#include<chrono>
#include <array>
using namespace std;
using namespace chrono;

void HOME(int arr[], int n){
    int i, key, j;
    for(i=1; i<n; i++){
        key = arr[i];
        j = i - 1;
        while(j>=0 && arr[j]>key){
            arr[j+1] = arr[j];
            j--;
        }
        arr[j+1] = key;
    }
}
```

b) Analisa Big O

Terdapat fungsi **HOME** yang melakukan pengurutan array dari kecil ke besar. Apabila ditemukan elemen yang tidak urut, maka akan masuk pada while statement yang melakukan pertukaran elemen array

- Baris 9, 11, 12, 14, 15, dan 17 merupakan konstan sehingga memiliki O(1)
- Baris 10 memiliki loop menggunakan for statement dan iterasi i++ sehingga memiliki O(n)
- Baris 13 memiliki loop menggunakan while statement dan iterasi j— sehingga memiliki O(n)

c) Analisa kasus

Isi dari array akan mempengaruhi kompleksitas waktu. Jika ditemukan elemen [j] lebih besar dari [j+1] maka akan dieksekusi while loop yang melakukan pertukaran. While loop memiliki O(n) sehingga akan ada perbedaan signifikan pada kasus

- Bestcase → Array dengan elemen urut dari kecil ke besar, tidak dieksekusi while loop sama sekali → O(n)
- Worstcase → Array dengan elemen urut dari besar ke kecil, dieksekusi while loop sebanyak n-1 kali → $O(n^2)$
- Averagecase → Array dengan elemen yang acak, while loop mengikuti elemen → $O(n^2)$

d) Testing

Pada test ini digunakan n dengan initial value 999, batas n<=10000, dan iterasi n=n+1000

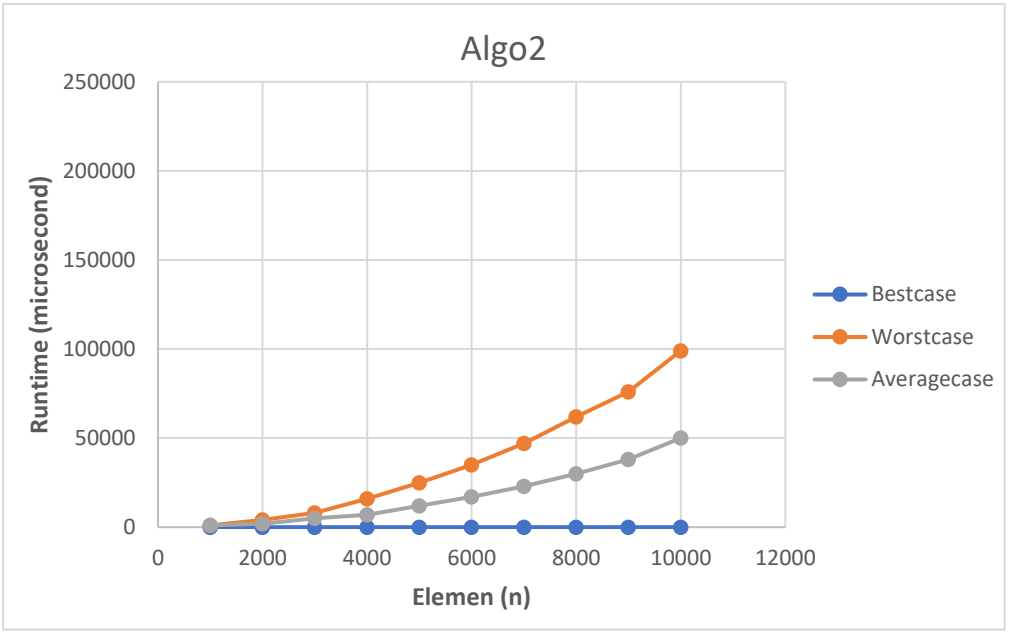
```cpp
int main(void){

    cout << "\tBestcase\tWorstcase\tAveragecase" << endl;
    for( int n=999; n<=10000; n+=1000 ){
        int* best = new int[n];
        int* worst = new int[n];
        int* avg = new int[n];

        for( int i=0; i<n; i++ ){
            best[i] = i;
            worst[i] = n-i;
            avg[i] = rand();
        }
```

```
34
35          auto best_t0 = high_resolution_clock::now();
36          HOME(best, n);
37          auto best_t1 = high_resolution_clock::now();
38          auto best_dt = best_t1-best_t0;
39          long long best_dtms = duration_cast <microseconds> (best_dt).count();
40
41          auto worst_t0 = high_resolution_clock::now();
42          HOME(worst, n);
43          auto worst_t1 = high_resolution_clock::now();
44          auto worst_dt = worst_t1-worst_t0;
45          long long worst_dtms = duration_cast <microseconds> (worst_dt).count();
46
47          auto avg_t0 = high_resolution_clock::now();
48          HOME(avg, n);
49          auto avg_t1 = high_resolution_clock::now();
50          auto avg_dt = avg_t1-avg_t0;
51          long long avg_dtms = duration_cast <microseconds> (avg_dt).count();
52
53          cout << "n=" << n << "\t" << best_dtms << "\t\t" << worst_dtms << "\t\t" << avg_dtms << endl;
54      }
```

```
D:\Kuliah\Semester 2\Struktur Data & Algoritma\prak\algoritma-2.exe
        Bestcase        Worstcase       Averagecase
n=999   0               1000            1000
n=1999  0               4001            2000
n=2999  0               8000            5001
n=3999  0               16003           7001
n=4999  0               25006           12002
n=5999  0               35006           17003
n=6999  0               47010           23005
n=7999  0               62013           30006
n=8999  0               76016           38008
n=9999  0               99021           50013

--------------------------------
Process exited after 0.6995 seconds with return value 0
Press any key to continue . . .
```

Note: runtime dari bestcase bernilai sangatlah kecil yang mana masih dianggap 0 dalam skala mikrosecond

e) Grafik



3. Algoritma-3.cpp

a) Source code:

algoritma-1.cpp | algoritma-2.cpp | algoritma-3.cpp

```cpp
#include<iostream>
#include<time.h>
#include<chrono>
#include <array>
#include <string>
using namespace std;
using namespace chrono;

int eraseAT(string str) {
    string acc;
    bool a, b;
    int x = 0;
    if (str.length() == 0)
        return 0;
    for (int i = 0; i < str.length(); i++) {
        if (str[i] == '[' || str[i] == ':' || str[i] == '|') {
            if (str[i] == '[') {
                acc.push_back(str[i]);
                a = true;
            }
            else if (str[i] == ':' && a) {
                if (str[i] == ':' && acc[acc.length() - 1] == ':') {
                    acc.pop_back();
                    b = false;
                }
                else {
                    acc.push_back(str[i]);
                    b = true;
                }
            }
            else if (str[i] == '|' && b) {}
            else x++;
        }
        else if (str[i] == ']') {
            if (acc.length() == 0)
                x++;
            else {
                acc.pop_back();
                a = false;
            }
        }
        else x++;
    }
    if (acc.length() == 0) return x;
    else return -1;
    for (int i = 0; i < str.length(); i++)
        for (int j = 0; j < str.length(); j++)
            for (int k = 0; k < str.length(); k++)
                str[i] = str[k];
    return 0;
}
```

b) Analisa Big O

Terdapat fungsi **eraseAT** yang menghitung jumlah karakter dari sebuah string dengan pengecualian pada empat karakter yaitu “[“, “:”, “|”, dan “]” yang memiliki mekanisme khusus. Mekanisme tersebut didukung oleh variabel pendukung a dan b yang bertipe boolean.

- Baris 15 memiliki loop menggunakan for statement dan iterasi i++ sehingga memiliki O(n)
- Selain loop, semua merupakan konstan sehingga memiliki O(1)

Terdapat perintah yang *unreachable* yaitu pada baris 46-50 dikarenakan baris tersebut terletak sesudah if else statement yang keduanya melakukan return.

c) Analisa kasus

Fungsi **eraseAT** memiliki perbedaan T(n) yang sangat kecil sehingga fungsi **eraseAT** dapat dianggap tidak memiliki kasus bestcase, worstcase, dan averagecase

- Semua kasus → O(n)

d) Testing

Untuk men-*generate* string dengan karakter yang acak, dibutuhkan fungsi pendukung yaitu fungsi **randchar**. Karakter yang akan dirandom adalah lowercase alphabet ditambah dengan "[", ":", "|", dan "]" (total ada 31 karakter)

```
53  char randchar(){
54      const array <char, 31> listchar = {'[', ':', '|', ']', 'a', 'b', 'c', 'd', 'e', 'f', 'g', 'h', 'i', 'j',
55      'k', 'l', 'm', 'n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'x', 'y', 'z', ' '};
56
57      return listchar[ rand()%31 + 1 ];
58
59  }
```

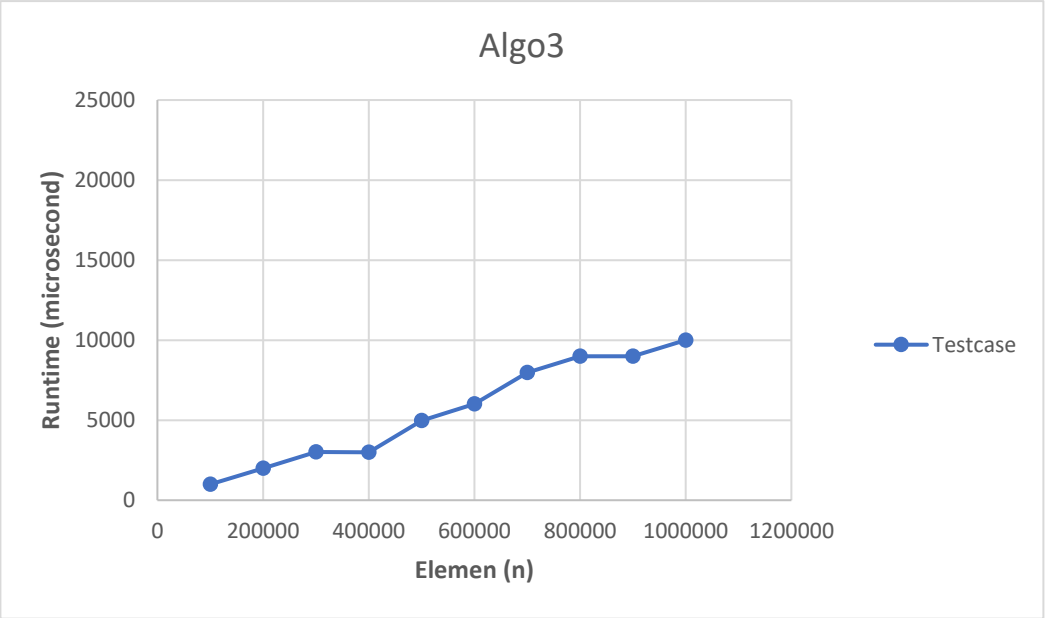
Pada test ini digunakan n dengan initial value 100000, batas n<= 1000000, dan iterasi n=n+100000

```
60
61  int main(void){
62
63      cout << "\t\tTestcase" << endl;
64      for( int n=100000; n<=1000000; n+=100000 ){
65          string str;
66
67          for( int i=0; i<n; i++ ){
68              str.push_back(randchar() );
69          }
70
71          auto t0 = high_resolution_clock::now();
72          int a = eraseAT(str);
73          auto t1 = high_resolution_clock::now();
74          auto dt = t1-t0;
75          long long dtms = duration_cast <microseconds> (dt).count();
76
77
78          cout << "n=" << n << "\t" << dtms << endl;
79      }
80  }
```

```
if (acc.length() == 0) return x;
```

```
D:\Kuliah\Semester 2\Struktur Data & Algoritma\prak\algoritma-3.exe
                Testcase
n=100000        993
n=200000        2000
n=300000        3006
n=400000        2994
n=500000        4982
n=600000        6006
n=700000        7982
n=800000        8987
n=900000        9002
n=1000000       10002

--------------------------------
Process exited after 0.3505 seconds with return value 0
Press any key to continue . . .
```

e) Grafik



Kesimpulan

Algoritma-1 dan Algoritma-2 memiliki task yang sama yaitu mengurutkan array dari kecil ke besar. Dapat dilihat bahwa Algoritma-2 memiliki runtime yang lebih cepat daripada Algoritma-1 sehingga dapat disimpulkan bahwa Algoritma-2 lebih bagus daripada Algoritma-1