MII1201 | Tutorial Pemrograman 3
Subprogram, Sorting, dan Searching

Kelas : PRG-8
Asisten : Chrystian {@mail.ugm.ac.id}

November 14, 2020

Dokumen dan code dapat diunduh melalui : https://github.com/Chrysophyt/Tutor_PRG8

1 Subprogram

Secara hasil eksekusi, Subprogram dapat dibagi menjadi 2 :

1. Procedural
2. Function

Procedural adalah Subprogram yang mengeksekusikan command/ algoritma secara bertahap, sedangkan Function sama tetapi biasanya mengembalikan *value*. Dalam C++ semua Subprogram adalah suatu Function, apabila Subprogram tidak mengembalikan *value* dapat menggunakan *void* type. Dalam Object Oriented Programming Function secara umumnya disebut dengan Class method.

1.1 Function Syntax

```
functionReturnType functionName(dataType Params1, .., dataType ParamsN){
    ...
}
```

Untuk functionReturnType dapat menggunakan void bila tidak *return* atau mengembalikan data. Apabila fungsi mengembalikan suatu float maka functionReturnType adalah float. dataType tidak harus tipe data primitive, C++ dapat mengembalikan array, struct, class, pointer, dsb.

Untuk functionName C++ dapat menggunakan camelCase atau snake_case Convention. Untuk fungsi pada camelCase biasa berawal dengan huruf awal lowercase dan apabila ada spasi akan dihapus dan huruf selanjutnya menjadi uppercase. Untuk snake_case semua karakter lowercase dan spasi diubah menjadi underscore. Contoh : Calculate Data Mean

```
//camelCase
float calculateDataMean(float *data, int nArraySize){
    ...
}
//snake_case
float calculate_data_mean(float *data, int n_array_size){
    ...
}
```

1.2 Function by Call

Ada 2 cara data masuk kedalam fungsi, apabila data di*copy* dan tidak mengubah data awal dinamakan Call by Value, sedangkan bila data langsung diubah dalam fungsi maka dinamakan Call by Reference. Pada umumnya fungsi menggunakan Call by Value, untuk menggunakan Call by Reference biasa memakai pointer, reference pada C++.

1.3 Example

```
#include<iostream>
using namespace std;

void printArray(float *data, int nArraySize){
    for(int i = 0; i < nArraySize; i++){
        cout << data[i]<<" ";
    }
    cout << "\n";
}

float calculateDataMean(float *data, int nArraySize){
    // *(data + i) = data[i], a pointer is an array !
    // data termasuk Call by Reference, nArraySize Call by Value
    float sum = 0;
    for(int i = 0; i < nArraySize; i++){
        sum += data[i];
    }
    return (sum/nArraySize);
}

void normalizeArray(float *data, int nArraySize){
    float xMin = data[0];
    float xMax = data[0];

    //Cari Minimum dan Maximum
    for(int i = 0; i < nArraySize; i++){
        if(xMin > data[i]){
            xMin = data[i];
        }
        if(xMax < data[i]){
            xMax = data[i];
        }
    }
    //Apply Normalize Function
    for(int i = 0; i < nArraySize; i++){
        data[i] = (data[i] - xMin)/(xMax-xMin);
    }

}

int main(){
    int n = 4;
    float data[n] = {3,4,5,6};

    printArray(data, n);
    cout << "Avg : "<< calculateDataMean(data, n)<<"\n";
    cout << "Normalizing Array : "<<"\n";
    normalizeArray(data, n);
    printArray(data, n);

}
    // Hasil
```

```
// 3 4 5 6
// Avg : 4.5
// Normalizing Array :
// 0 0.333333 0.666667 1
```

2 Sorting

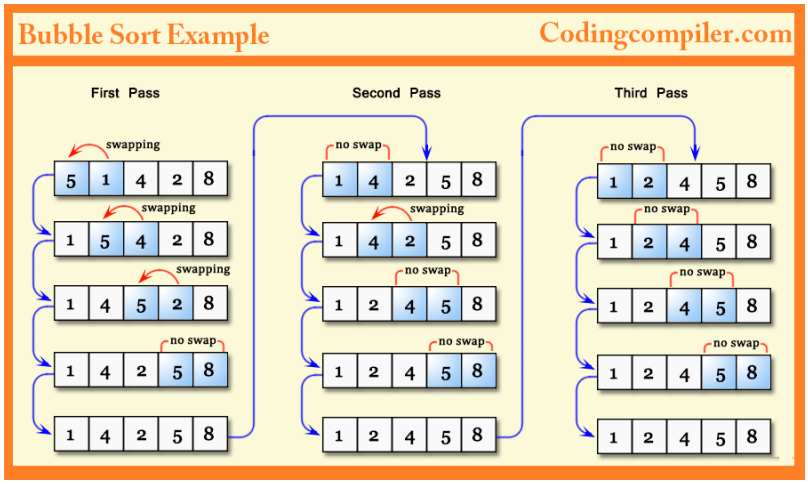
Array Sorting Algorithms

Algorithm	Time Complexity			Space Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)
Heapsort	Ω(n log(n))	θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	θ(n log(n))	O(n log(n))	O(n)

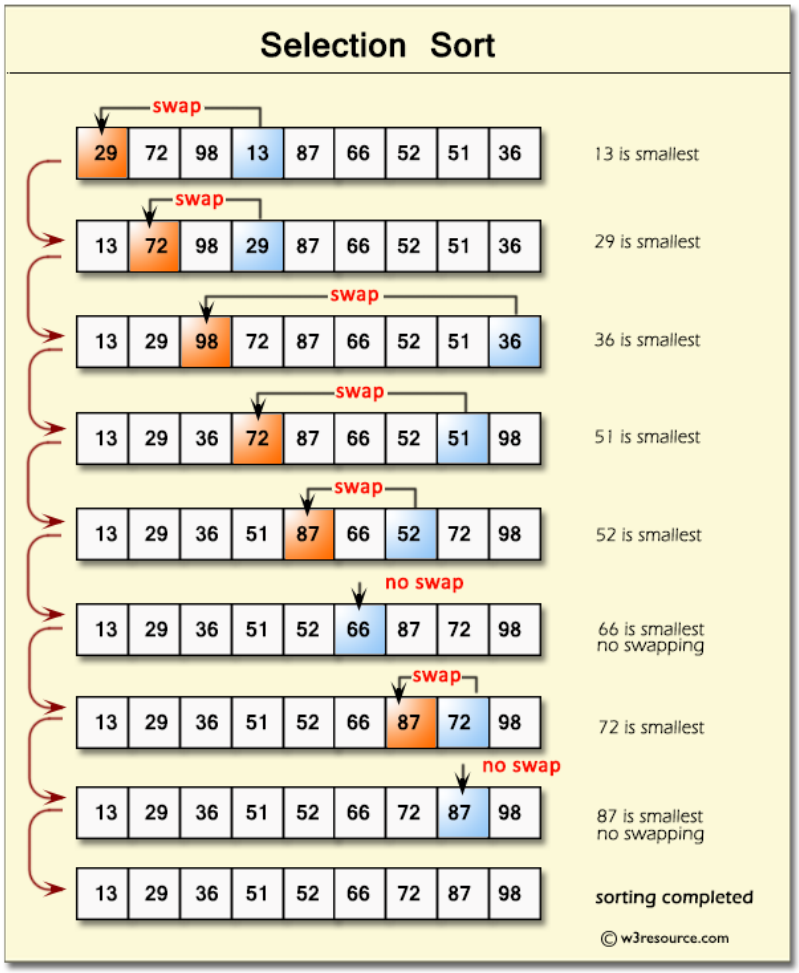
Secara umum Quick Sort adalah metode algoritma terbaik untuk mensortir unordered data, secara praktik komputer dapat menukar data lebih cepat dari pada membuat dan salin ulang data seperti algoritma Merge Sort. Worst Case Quick Sort jarang sekali terjadi, kasus dapat terjadi jika :

- Array sudah tersortir berurut.
- Array sudah tersortir secara urutan terbalik.
- Semua elemen sama.

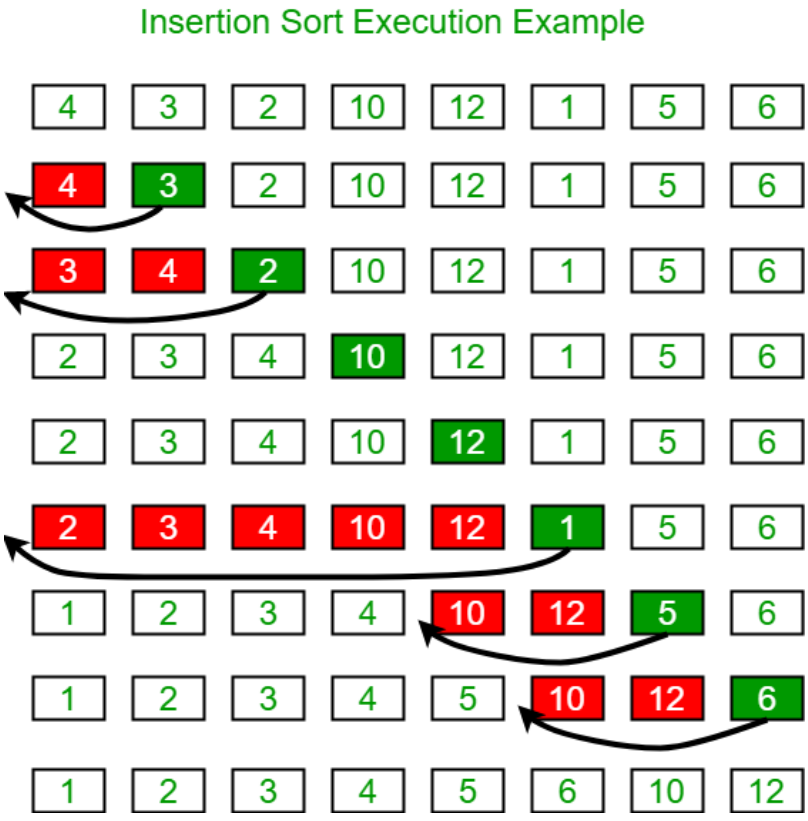
2.1 Bubble Sort



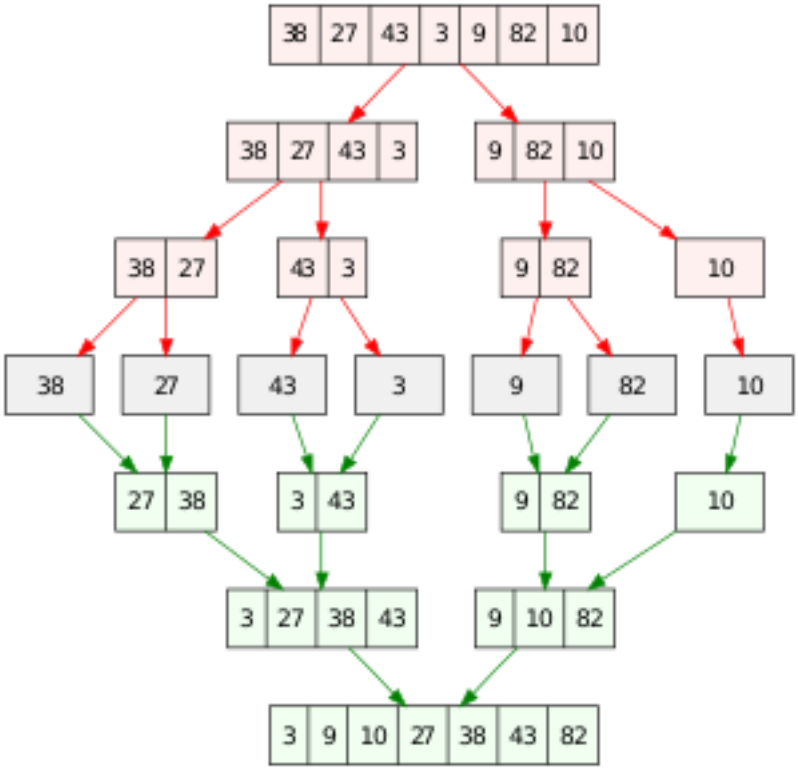
2.2 Selection Sort



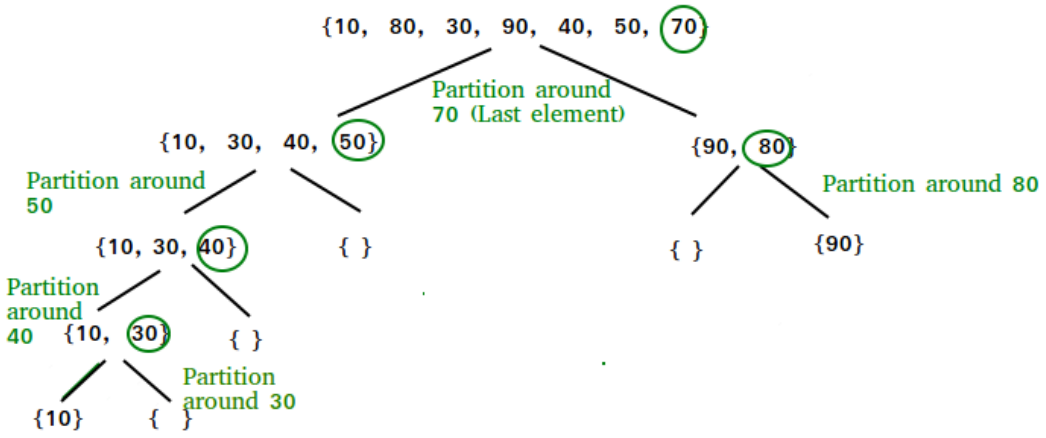
2.3 Insertion Sort



2.4 Merge Sort

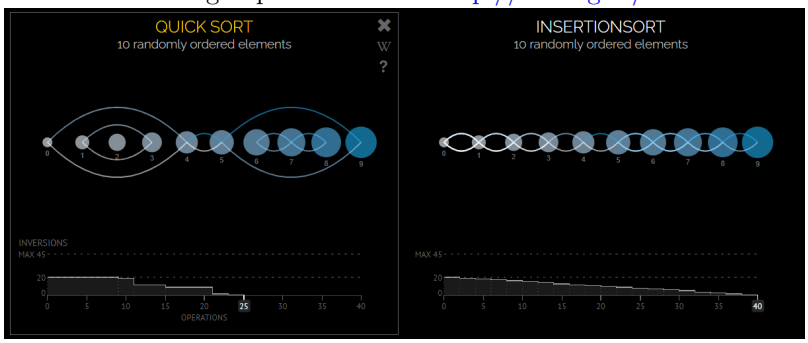


2.5 Quick Sort



2.6 Visualization

Visualisasi Sorting dapat dilihat di : http://sorting.at/

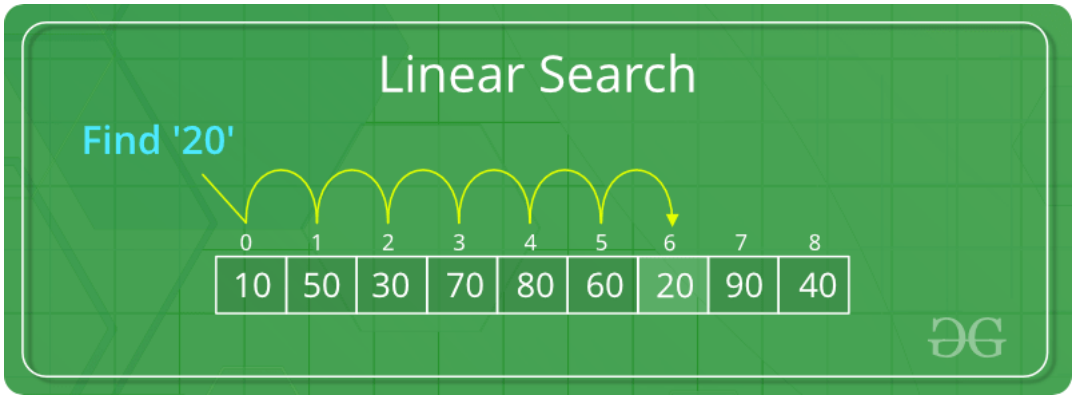


2.7 UAS 2018 No.3

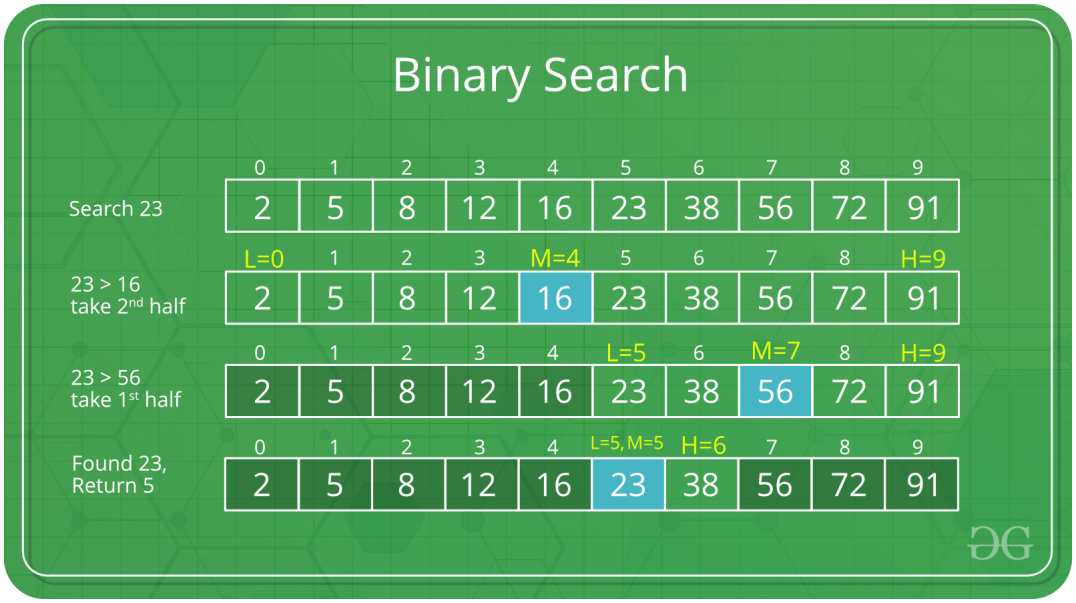
Diketahui sekumpulan bilangan sebagai berikut:
25, 73, 41, 30, 58, 64, 98, 13, 87, 91, 17, 76, 28, 45, 56
Dengan menggunakan data input diatas, tuliskan urutan data sampai terurut menaik dan hitung berapa kali operasi pertukaran (*swapping*) dilakukan jika digunakan algoritme Insertion Sort ?

3 Searching

3.1 Linear Search



3.2 Binary Search



3.3 Kekurangan

Permasalahan solusi Searching terbaik sangat tergantung kasus demi kasus penggunaannya. Apabila jarang dilakukan searching dan data relatif besar, jauh lebih efesien menggunakan Linear Search, tetapi misal searching harus dilakukan banyak sekali, alasan tersebut dapat menjadi justifikasi melakukan additional computation sorting untuk melakukan Binary Search.

3.3.1 Linear Search

Bila data yang ingin ditemukan tidak ada pada elemen maka Linear Search akan mendapatkan Worst Case. Sangat tidak disarankan melakukan Linear Search apabila memiliki kemungkinan tinggi elemen tidak ada dalam suatu Array. Complexity Time terburuk dibanding metode lain.

3.3.2 Binary Search

Isi dari Array tersebut harus selalu urut, dimana dalam beberapa kasus (seperti data yang terupdate realtime) sangat sulit dipenuhi. Binary Search lebih kompleks dan sangat membuang-buang waktu (*overkill*) apabila data hanya memiliki elemen sedikit.