

**Jawaban Studi Kasus 1**

**1. Program C++**

#include <iostream>

using namespace std;

void Merge(int \*a, int low, int high, int mid){

    int i, j, k, temp[high-low+1];

    i = low;

    k = 0;

    j = mid + 1;

    while (i <= mid && j <= high){

        if (a[i] < a[j]){

            temp[k] = a[i];

            k++;

            i++;

        }

        else{

            temp[k] = a[j];

            k++;

            j++;

        }

    }

    while (i <= mid){

        temp[k] = a[i];

        k++;

        i++;

    }

    while (j <= high){

        temp[k] = a[j];

        k++;

        j++;

    }

    for (i = low; i <= high; i++){

        a[i] = temp[i-low];

    }

}

void MergeSort(int \*a, int low, int high){

    int mid;

    if (low < high){

        mid=(low+high)/2;

        MergeSort(a, low, mid);

        MergeSort(a, mid+1, high);

        Merge(a, low, high, mid);

    }

}

int main(){

    int n, i;

    int arr[n];

    cout<<"\nMasukkan Jumlah Data : ";cin>>n;

    for(i = 0; i < n; i++){

        cout<<"Masukkan elemen ke-"<<i+1<<": ";

        cin>>arr[i];

    }

    MergeSort(arr, 0, n-1);

    cout<<"\nData Terurut: ";

    for (i = 0; i < n; i++)

        cout<<" "<<arr[i];

    return 0;

}

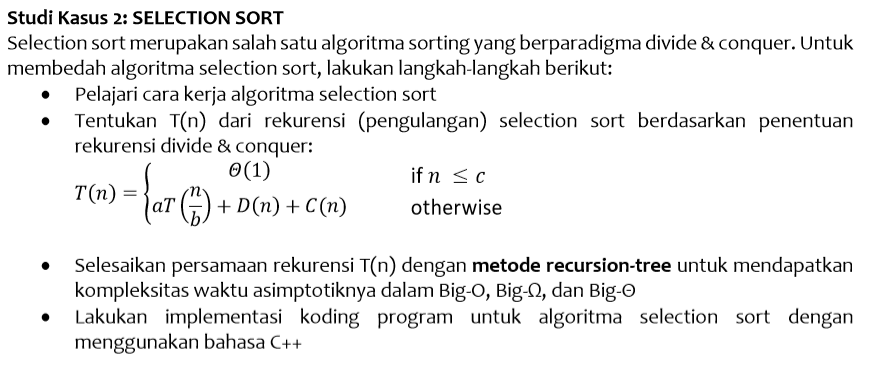
**2. Kompleksitas waktu**



Input n = 20

Durasi waktu yang dibutuhkan untuk 20 input: 61572475 ms = 61.572475 s

Big-O = Big-Ω = Big-θ = (n log n)

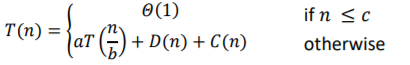


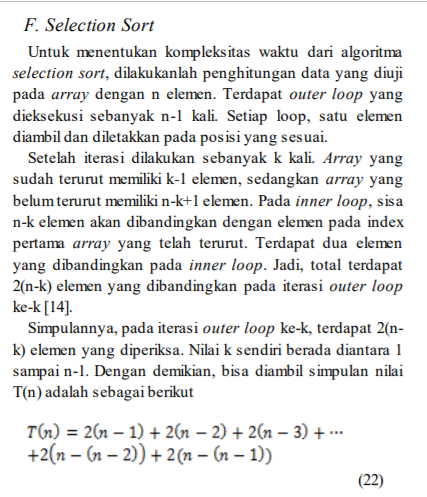
**Jawaban Studi Kasus 2**

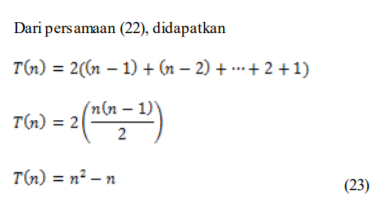
**1. Cara Kerja selection sort**

* Melakukan pengecekan dimulai dari data pertama hingga data ke-n.
* Menentukan data dengan indeks minimum (jika acending) atau maksimum (jika descending) dalam sebuah data tersebut.
* Menukarkan data dengan indeks minimum (jika ascending) atau maksimum (jika descending) dengan bilangan pertama (i = 1) dari data tersebut.
* Mengulangi langkah di atas untuk sisa data bilangan berikutnya (i = i +1) sampai didapatkan urutan yang sesuai.

**2. Tentukan T(n) dari selection sort**







**3. Kompleksitas waktu**

Big-O = Big-Ω = Big-θ = n2

**4. Program C++**

#include <iostream>

using namespace std;

int main(){

    int n,x[100],imaks,temp;

    cout << "Masukkan Jumlah Data : ";cin >> n;

    for (int i = 0; i < n; i++){

        cout << "Bilangan ke - " << i+1 << " : ";

        cin >> x[i];

    }

    cout << "\nSebelum di Sorting : ";

    for (int i = 0; i < n; i++)

        cout << x[i] << " ";

        cout << endl;

    for (int i = n-1; i >= 1; i--){

        imaks = 0;

        for (int j = 1; j <= i; j++){

            if (x[j] > x[imaks])

                imaks = j;

        }

        temp = x[i];

        x[i] = x[imaks];

        x[imaks] = temp;

    }

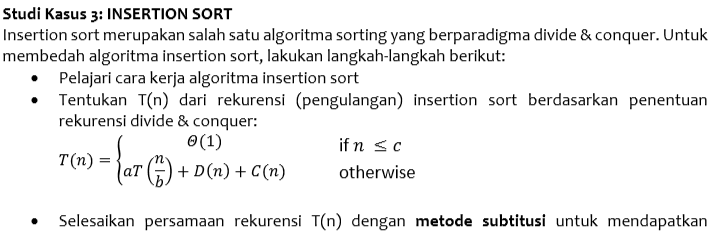
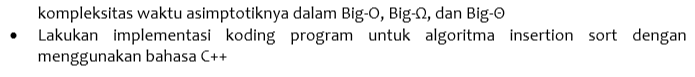
    cout << "Setelah di Sorting : ";

    for (int i = 0; i < n; i++)

        cout << x[i] << " ";

    return 0;

}

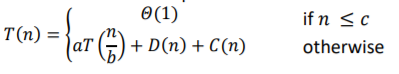


**Jawaban Studi Kasus 3**

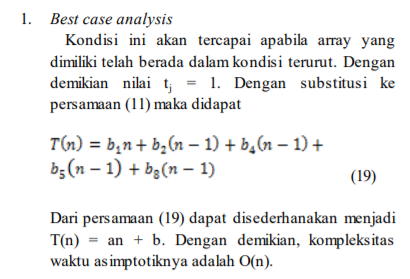
**1. Cara Kerja Insertion sort**

* Pengecekan mulai dari data ke-1 sampai data ke-n
* Bandingkan data ke-I ( I = data ke-2 s/d data ke-n )
* Bandingkan data ke-I tersebut dengan data sebelumnya(I-1), Jika lebih kecil maka data tersebut dapat disisipkanke data awal sesuai dgn posisisi yg seharusnya
* Lakukan langkah 2 dan 3 untuk bilangan berikutnya( I= I+1 ) sampai didapatkan urutan yg optimal.

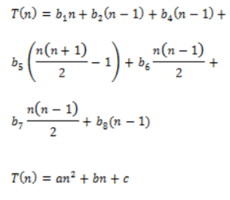
**2. T(n) dari insertion sort**



Best Case :



Worst Case:

Nilai waktu asimptotiknya adalah O(n2)

**3. Kompleksitas waktu**

Big-O = n

Big-Ω = Big-θ = n2

**4. Program C++**

#include <iostream>

using namespace std;

int main(){

    int n,x[100],j;

    int insert;

    cout << "Masukkan Jumlah Data : "; cin >> n;

    for (int i = 0; i < n; i++){

        cout << "Bilangan ke - " << i+1 << " : ";

        cin >> x[i];

    }

    cout << "\nSebelum di Sorting : ";

    for (int i = 0; i < n; i++)

        cout << x[i] << " ";

        cout << endl;

    for (int i = 1; i < n; i++){

        insert = x[i];

        j = i-1;

        while ((j >= 0) && (x[j] > insert)){

            x[j+1] = x[j];

            j--;

        }

        x[j+1] = insert;

    }

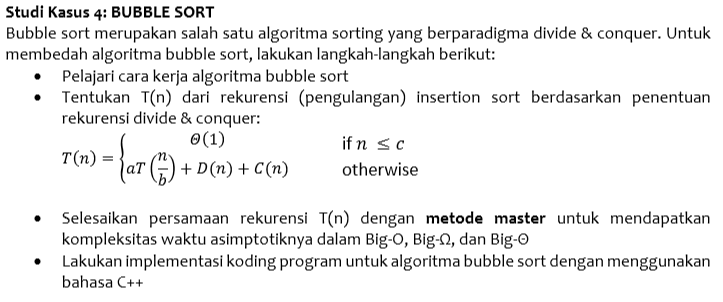
    cout << "Setelah di Sorting : ";

    for (int i = 0; i < n; i++)

        cout << x[i] << " ";

    return 0;

}

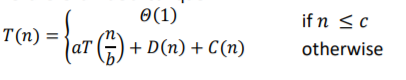


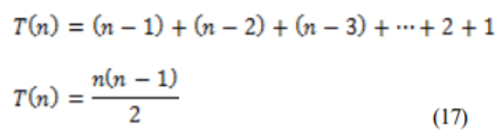
**Jawaban Studi Kasus 4**

**1. Cara Kerja Bubble Sort**

* Pengecekan mulai dari data ke satu sampai data ke-n
* Bandingkan data ke-n dengan data sebelumnya (n-1)
* Jika lebih kecil maka pindahkan bilang tersebut dengan bilang yang ada didepannya (sebelumnya) satu persatu (n-1,n-2,n-3,….dts)
* Jika lebih besar maka tidak terjadi permindahan 5.Ulang langkah 2 dan 3 s/d sort optimal.

**2. T(n) dari bubble sort**





**3. Kompleksitas waktu**

Big-O = n

Big-Ω = Big-θ = n2

**4. Program C++**

#include <iostream>

using namespace std;

int main(){

    int arr[100],n,temp;

    cout<<"\nMasukkan Jumlah Data : ";cin>>n;

    for(int i=0;i<n;++i){

        cout<<"Bilangan ke-"<<i+1<<" : ";cin>>arr[i];

    }

    for(int i=1;i<n;i++){

        for(int j=0;j<(n-1);j++){

            if(arr[j]>arr[j+1]){

                temp=arr[j];

                arr[j]=arr[j+1];

                arr[j+1]=temp;

            }

        }

    }

    cout<<"\nOutput Bubble Sort : ";

    for(int i=0;i<n;i++){

        cout<<" "<<arr[i];

    }

    return 0;

}