SORTING

Sorting adalah proses menyusun elemen-elemen dengan tata urut tertentu dan proses tersebut terimplementasi dalam bermacam aplikasi.

Macam-macam sorting:

1. **Bubble Sort**

Metode:

Bubble sort (metode gelembung) adalah metode/algoritma pengurutan dengan cara melakukan penukaran data dengan tepat disebelahnya secara terus menerus sampai bisa dipastikan dalam satu iterasi tertentu tidak ada lagi perubahan. Jika tidak ada perubahan berarti data sudah terurut. Disebut pengurutan gelembung karena masing-masing kunci akan dengan lambat menggelembung ke posisinya yang tepat.

Metode ini terinspirasi oleh gelembung sabun yang berada dipermukaan air. Karena berat jenis gelembung sabun lebih ringan daripada berat jenis air, maka gelembung sabun selalu terapung ke atas permukaan.

Kelebihan:

* Metode buble sort merupakan metode yang paling simpel karena mudah dipahami algoritmanya
* Definisi terurut terdapat dengan jelas dalam algoritma.
* Cocok untuk pengurutan data dengan elemen kecil telah terurut

Kekurangan:

Meskipun simpel metode bubble sort merupakan metode pengurutan yang paling tidak efisien. Pada saat mengurutkan data yang sangat besar akan mengalami kelambatan luar biasa ketika data yang diolah. Kelemahan lainnya adalah jumlah pengulangan akan tetap sama jumlahnya walaupun data sesungguhnya sudah cukup terurut. Hal ini disebabkan setiap data dibandingkan dengan setiap data yang lain untuk menentukan posisinya.

Codingan (dari kelompok 1):

#include<iostream>

using namespace std;

void swap(int &x, int &y){

int temp=x;

x=y;

y=temp;

}

int main(){

int n;

cout<<"Masukkan Jumlah Data= ";

cin>>n;

int a[n];

cout<<"Masukkan Elemen-Elemen Data = ";

for(int i =0;i<n;i++){

cin>>a[i];

}

cout<<"Data Yang Sudah Di Urutkan = ";

for(int i=n-1;i>0;i--){

for(int j=0;j<i;j++){

if(a[j]>a[j+1]){

swap(a[j],a[j+1]);

}

}

}

for(int j=0;j<n;j++){

cout<<a[j]<<" ";

}

}

1. **Selection Sort**

Metode:

Selection Sort merupakan salah satu algoritma pengurutan yang dasarnya adalah melakukan beberapa kali pass untuk melakukan penyeleksian elemen struktur data.

Untuk sorting ascending (menaik), elemen yang paling kecil di antara elemen-elemen yang belum urut disimpan indeksnya, kemudian dilakukan pertukaran nilai elemen dengan indeks yang disimpan tersebut dengan elemen yang paling depan yang belum urut. Sebaliknya, untuk sorting descending (menurun), elemen yang paling besar yang disimpan indeksnya kemudian ditukar.

Selection Sort bekerja sebagai berikut:

* Mencari nilai minimum (jika ascending) atau maksimum (jika descending) dalam sebuah list
* Menukarkan nilai ini dengan elemen pertama list
* Mengulangi langkah di atas untuk sisa list dengan dimulai pada posisi kedua

Secara efisien kita membagi list menjadi dua bagian yaitu bagian yang sudah diurutkan, yang didapat dengan membangun dari kiri ke kanan dan dilakukan pada saat awal, dan bagian list yang elemennya akan diurutkan.

Kelebihan:

* Algoritma ini sangat rapat dan mudah untuk diimplementasikan.
* Operasi pertukarannya hanya dilakukan sekali saja.
* Waktu pengurutan dapat lebih ditekan.
* Mudah menggabungkannya kembali.
* Kompleksitas selection sort relatif lebih kecil.

Kekurangan:

* Sulit untuk membagi masalah.

Codingan:

#include<iostream>

using namespace std;

int main(){

typedef int angka[100];

int i,j,n,temp,minindex;

angka x;

cout<<"Program Selection Sort"<<endl<<endl;

cout<<"Masukkan banyak data: ";

cin>>n;

cout<<endl<<"Masukkan data:\n";

for(i=0; i<n; i++){

cout<<"x["<<i+1<<"] = ";

cin>>x[i];

}

cout<<endl<<"Data sebelum di sort: ";

for(i=0; i<n;i++){

cout<<x[i]<<" ";

}

for(i=0; i<n-1; i++){

minindex=i;

for(j=i+1; j<n; j++){

if(x[minindex]>x[j]){

minindex=j;

}

}

temp=x[i];

x[i]=x[minindex];

x[minindex]=temp;

}

cout<<endl<<"Data setelah di sort: ";

for(i=0; i<n; i++){

cout<<x[i]<<" ";

}

}

1. **Insertion Sort**

Metode:

Insertion sort adalah metode pengurutan dengan cara menyisipkan elemen larik pada posisi yang tepat.

Macam- macam metode insertion sort:

* Langsung (*Straight Insertion Sort*)
* Metode Penyisipan Biner (*Binary Insertion Sort*)

Metode pengurutan dengan algoritma penyisipan biner (*binary insertion sort*) memperbaiki metode pengurutan dengan algoritma penyisipan langsung dengan melakukan proses perbandingan yang lebih sedikit sehingga proses pengurutan lebih cepat.

Metode penyisipan biner melakukan proses perbandingan dengan membagi dua bagian data dari posisi 0 sampai dengan *i*-1 yang disebut dengan bagian kiri dan kanan. Apabila data pada posisi ke *i* berada pada jangkauan kiri maka proses perbandingan dilakukan hanya pada bagian kiri dan menggeser posisi sampai *i*.

Kelebihan:

* Sederhana dalam penerapannya.
* Cepat dalam data yang kecil dan yang sebagian sudah terurut.
* Jika list sudah terurut atau sebagian terurut maka Insertion Sort akan lebih cepat dibandingkan dengan Quick Sort.
* Lebih mangkus dibanding Bubble Sort dan Selection Sort.
* Loop dalam pada Insertion Sort sangat cepat, sehingga membuatnya salah satu algoritma pengurutan tercepat pada jumlah elemen yang sedikit.

Kekurangan:

* Banyaknya operasi yang diperlukan dalam mencari posisi yang tepat untuk elemen larik.
* Untuk larik yang jumlahnya besar ini tidak praktis.
* Jika list terurut terbalik setiap eksekusi dari perintah harus memindai dan mengganti seluruh bagian sebelum menyisipkan elemen berikutnya.

Codingan (dari kelompok 3):

#include<iostream>

using namespace std;

typedef int larik[10];

void swap(int &x,int &y){

int temp;

temp=x;

x=y;

y=temp;

}

void insertionSort(larik x,int n) {

for(int i=1;i<n;i++) {

for(int j=i;j>=1;j--){

if (x[j]<x[j-1]){

swap(x[j],x[j-1]);

}

else{

break;

}

}

}

}

void banyakData(int& nData){

cout<<"Banyak Data: ";

cin>>nData;

}

void isiLarik(larik& x, int nData){

char nama[20];

for(int i=0;i<nData;i++) {

cout<<"Data ke-["<<i+1<<"]: ";

cin>>x[i];

}

}

void InsertionSort(larik x, int nData){

cout<<"\nKondisi Awal:\n";

for(int i=0;i<nData;i++) {

cout<<x[i]<<" ";

}

cout<<"\nSetelah Insertion Sort:\n";

insertionSort(x, nData);

for(int i=0;i<nData;i++) {

cout<<x[i]<<" ";

}

}

int main(){

larik x;

int nData;

banyakData(nData);

isiData(x, nData);

InsertionSort(x, nData);

}

1. **Shell Sort**

Metode:

Metode ini dikembangkan oleh Donald L. Shell pada tahun 1959. Metode ini disebut juga dengan metode pertambahan menurun (diminishing increment). Metode ini mengurutkan data dengan cara membandingkan suatu data dengan data lain yang memiliki jarak tertentu, kemudian dilakukan penukaran bila diperlukan.

Proses pengurutan dengan metode Shell dapat dijelaskan sebagai berikut:

Pertama-tama adalah menentukan jarak mula-mula dari data yang akan dibandingkan, yaitu N / 2. Data pertama dibandingkan dengan data dengan jarak N / 2. Apabila data pertama lebih besar dari data ke N / 2 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan dengan jarak yang sama yaitu N / 2. Demikian seterusnya sampai seluruh data dibandingkan sehingga semua data ke-j selalu lebih kecil daripada data ke-(j + N / 2).  
Pada proses berikutnya, digunakan jarak (N / 2) / 2 atau N / 4. Data pertama dibandingkan dengan data dengan jarak N / 4. Apabila data pertama lebih besar dari data ke N / 4 tersebut maka kedua data tersebut ditukar. Kemudian data kedua dibandingkan dengan jarak yang sama yaitu N / 4. Demikianlah seterusnya hingga seluruh data dibandingkan sehingga semua data ke-j lebih kecil daripada data ke-(j + N / 4).  
Pada proses berikutnya, digunakan jarak (N / 4) / 2 atau N / 8. Demikian seterusnya sampai jarak yang digunakan adalah 1.

Kelebihan:

* Algoritma ini sangat rapat dan mudah untuk diimplementasikan.
* Operasi pertukarannya hanya dilakukan sekali saja.
* Waktu pengurutan dapat lebih ditekan.
* Mudah menggabungkannya kembali.
* Kompleksitas selection sort relatif lebih kecil.

Kekurangan:

* Membutuhkan method tambahan.
* Sulit untuk membagi masalah.

Codingan (dari kelompok 4):

#include<iostream>

using namespace std;

typedef int larik [10];

void swap(int &x,int y){

int temp;

temp=x;

x=y;

y=temp;

}

void shellSort(int a[],int n){

int j;

//Pengurangan jarak/gap sebanyak 2

for(int gap=n/2;gap>0;gap/=2){

for(int i=gap;i<n;i++){

int temp=a[i];

for (j=i;j>=gap&&temp<a[j-gap];j-=gap){

a[j]=a[j-gap];

}

a[j] = temp;

}

}

}

void banyakData(int& nData){

cout<<"Banyak Data: ";

cin>>nData;

}

void isi(larik& x,int nData){

for(int i=0;i<nData;i++){

cout<<"Data ke-["<<i+1<<"]: ";

cin>>x[i];

}

}

void cetakLarik(larik x, int nData){

cout<<"Kondisi Awal Larik: ";

for(int i=0;i<nData;i++){

cout<<x[i]<<" ";

}

cout<<endl;

cout<<"Setelah Shell Sort: ";

shellSort(x,nData);

for(int i=0;i<nData;i++){

cout<<x[i]<<" ";

}

}

int main(){

larik x;

int nData;

banyakData(nData);

isi(x,nData);

cetakLarik(x,nData);

}

1. **Quick Sort**

Quick sort adalah algoritma sorting yang berdasarkan pembandingan dengan metoda divide-and-conqueror. Disebut Quick Sort, karena algoritma ini mengurutkan dengan sangat cepat namun algoritma ini sangat kompleks dan diproses secara rekursif. Sangat memungkinkan untuk menulis algoritma yang lebih cepat untuk beberapa kasus khusus, namun untuk kasus umum, sampai saat ini tidak ada yang lebih cepat dibandingkan 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 irterasi pengurutan, pertama-tama sebuah elemen dipilih dari data,  kemudian elemen-elemen data akan diurutkan diatur sedemikian rupa.

Di bawah ini akan dijelaskan langkah-langkah strategi divide-and-conqueror:

* Pilih nilai pivot, Kita ambil nilai di tengah-tengah elemen sebagai sebagai nilaidaripivot tetapi bisa nilai mana saja.
* Partisi**,**  Atur ulang semua elemen sedemikian rupa, lalu semua elemen yang lebih rendah daripada pivot dipindahkan ke sebelah kiri dari array/list dan semua elemen yang lebih besar dari  pivot dipindahkan ke sebelah kanan dari array/list. Nilai yang sama dengan pivot  dapat diletakkan di mana saja dari array. Ingat, mungkin array/list akan dibagi dalam bagian yang tidak sama.
* Urutkan semua bagian (kiri/kanan), Aplikasikan algoritma quicksort secara rekursif pada bagian sebelah kiri dan kanan.

Kelebihan:

* Secara umum memiliki kompleksitas O(n log n).
* Algoritmanya sederhana dan mudah diterapkan pada berbagai bahasa pemrograman dan arsitektur mesin secara efisien.
* Dalam prakteknya adalah yang tercepat dari berbagai algoritma pengurutan dengan perbandingan, seperti merge sort dan heap sort.
* Melakukan proses langsung pada input (in-place) dengan sedikit tambahan memori.
* Bekerja dengan baik pada berbagai jenis input data (seperti angka dan karakter).

Kekurangan:

* Sedikit kesalahan dalam penulisan program membuatnya bekerja tidak beraturan (hasilnya tidak benar atau tidak pernah selesai).
* Memiliki ketergantungan terhadap data yang dimasukkan, yang dalam kasus terburuk memiliki kompleksitas O(n2).
* Secara umum bersifat tidak stable, yaitu mengubah urutan input dalam hasil akhirnya (dalam hal inputnya bernilai sama).
* Pada penerapan secara rekursif (memanggil dirinya sendiri) bila terjadi kasus terburuk dapat menghabiskan stack dan memacetkan program.

Codingan:

#include<iostream>

using namespace std;

void input(int (&a)[1000],int &n){

cout<<"Input banyaknya data: ";

cin>>n;

cout<<endl;

for(int i=0;i<n;i++){

cout<<"Masukkan data ke ["<<i+1<<"]= ";

cin>>a[i];

}

}

void urut(int (&a)[1000],int n){

int tukar,temp;

for(int i=1;i<=n;i++){

tukar=0;

for(int j=0;j<n-1;j++){

if(a[j+1]<a[j]){

temp=a[j];

a[j]=a[j+1];

a[j+1]=temp;

tukar=1;

}

}

}

}

void output(int (&a)[1000],int n){

urut(a,n);

cout<<"Setelah disortir menjadi = [ ";

for(int i=0;i<n;i++){

cout<<a[i]<<" ";

}

cout<<"]"<<endl<<endl;

}

int main(){

int ang[1000],n;

input(ang,n);

output(ang,n);

}

1. **Merge Sort**

Metode:

Merge sort merupakan algoritma pengurutan dalam ilmu komputer yang dirancang untuk memenuhi kebutuhan pengurutan atas suatu rangkaian data yang tidak memungkinkan untuk ditampung dalam memori komputer karena jumlahnya yang terlalu besar. Algoritma ini ditemukan oleh John von Neumann pada tahun 1945.  
  
Algoritma pengurutan data merge sort dilakukan dengan menggunakan cara divide and conquer yaitu dengan memecah kemudian menyelesaikan setiap bagian kemudian menggabungkannya kembali. Pertama data dipecah menjadi 2 bagian dimana bagian pertama merupakan setengah (jika data genap) atau setengah minus satu (jika data ganjil) dari seluruh data, kemudian dilakukan pemecahan kembali untuk masing-masing blok sampai hanya terdiri dari satu data tiap blok.  
  
Setelah itu digabungkan kembali dengan membandingkan pada blok yang sama apakah data pertama lebih besar daripada data ke-tengah+1, jika ya maka data ke-tengah+1 dipindah sebagai data pertama, kemudian data ke-pertama sampai ke-tengah digeser menjadi data ke-dua sampai ke-tengah+1, demikian seterusnya sampai menjadi satu blok utuh seperti awalnya. Sehingga metode merge sort merupakan metode yang membutuhkan fungsi rekursi untuk penyelesaiannya.

Kelebihan:

Kelebihan dari merge sort adalah waktu pengerjaan yang cepat dan efisien karena setiap list selalu dibagi bagi menjadi list yang lebih kecil, kemudian digabungkan lagi sehingga tidak perlu melakukan banyak perbandingan.

Kekurangan:

Kelemahan utama merge sort adalah algoritma ini membutuhkan kapasitas memori yang cukup besar karena dilakukan secara rekursif dan memakai dua elemen terpisah. Merge sort juga cukup sulit untuk diterapkan berkaitan dengan panjangnya instruksi yang diperlukan.

Codingan:

#include <iostream>

using namespace std;

int a[50];

void merge(int,int,int);

void merge\_sort(int bawah,int atas){

int tengah;

if(bawah<atas){

tengah=(bawah+atas)/2;

merge\_sort(bawah,tengah);

merge\_sort(tengah+1,atas);

merge(bawah,tengah,atas);

}

}

void merge(int bawah,int tengah,int atas){

int h,i,j,b[50],k;

h=bawah;

i=bawah;

j=tengah+1;

while((h<=tengah)&&(j<=atas)){

if(a[h]<=a[j]){

b[i]=a[h];

h++;

}

else{

b[i]=a[j];

j++;

}

i++;

}

if(h>tengah){

for(k=j;k<=atas;k++){

b[i]=a[k];

i++;

}

}

else{

for(k=h;k<=tengah;k++){

b[i]=a[k];

i++;

}

}

for(k=bawah;k<=atas;k++){

a[k]=b[k];

}

}

int main(){

int angka,i;

cout<<" MERGE SORT PROGRAM"<<endl;

cout<<"\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*"<<endl;

cout<<"Masukkan Banyaknya Angka yang Diinginkan: ";

cin>>angka;

cout<<endl;

cout<<"Masukkan Nilai dari "<< angka <<" Angka Tersebut: "<<endl;

for(i=1;i<=angka;i++){

cin>>a[i] ;

}

merge\_sort(1,angka);

cout<<endl;

cout<<"Hasil Pengurutan Menggunakan Merge Sort: "<<endl;

for(i=1;i<=angka;i++){

cout<<a[i]<<" ";

}

}

SEARCHING

Searching adalah mencari data yang dibutuhkan. Searching dalam pemrograman bisa dilakukan untuk mencari data yang ada di dalam memori komputer.

Macam-macam searching:

* 1. **Linear Search**

Metode:

Linear Search adalah metode pencarian sebuah data dari suatu kumpulan data dimana data dicari dari depan ke belakang atau dari awal sampai akhir data tanpa harus data tersebut terurut. Konsepnya yaitu dengan melakukan perbandingan data satu-persatu secara berurutan smapai data tersebut ditemukan ataupun tidak di temukan.

Proses:

* Pertama data melakukan perbandingan satu per satu secara berurutan dalam kumpulan data dengan data yang  dicari sampai data tersebut ditemukan atau tidak ditemukan.
* Pada dasarnya, pencarian ini hanya melakukan pengulangan data dari 1 sampai dengan jumlah data (n).
* Setiap pengulangan, dibandingkan data ke-i dengan data yang sedang dicari.
* Apabila data sama dengan yang dicari, berarti data telah berhasil ditemukan.Sebaliknya apabila sampai akhir melakukan pengulangan tidak ada data yang sama dengan yang dicari, berarti data tidak ada yang ditemukan.

Kelebihan:

* Relatif lebih cepat dan efisien untuk data yang memiliki jumlah terbatas
* Algoritma perogrammannya lebih sederhana

Kekurangan:

Kekurangannya jika data yang terdapat dalam suatu array itu sangat banyak, maka akan diperlukan waktu yang lebih lama untuk membandingkan data yang dicari dengan jumlah data yang sangat banyak dalam suatu array.

Codingan (dari kelompok 5):

#include <iostream>

using namespace std;

void input(int (&a)[100], int &n, int &c){

cout<<"Masukkan banyaknya data: ";

cin>>n;

for(int i=0;i<n;i++){

cout<<"Masukkan data ke ["<<i+1<<"] = ";

cin>>a[i];cout<<endl;

}

cout<<"Angka yang akan dicari: ";

cin>>c;cout<<endl;

}

void cari(int (&a)[100],int n,int c){

int found=0;

for(int i=0;i<n;i++){

if(c==a[i]){

found=1;

cout<<"Nilai yang dicari ditemukan di data ke-["<<i+1<<"] pada indeks ke-["<<i<<"]"<<endl;

}

}

if(found==0) cout<<"TIDAK ADA ANGKA YANG DICARI DI DALAM ARRAY DATA"<<endl;

}

void output(int (&a)[100],int n,int c){

cout<<"ARRAY DATA = [";

for(int i=0;i<n;i++){

cout<<a[i]<<" ";

}

cout<<"]"<<endl<<endl;

cari(a,n,c);

}

int main(){

int angka[100],n,c;

input(angka,n,c);

output(angka,n,c);

}

* 1. **Binary Search**

Metode:

Binary Search adalah metode pencarian sebuah data dari suatu kumpulan data, dimana kumpulan data tersebut harus berurutan dengan benar agar proses pencarian data bisa dilakukan. Dalam proses pencarian data pada metode ini data akan di bagi menjadi dua bagian untuk setiap tahap pencariannya.

Proses:

* Pertama pengambilan data dimulai  dari posisi 1 sampai  dengan posisi akhir (n)
* Selanjutnya mencari posisi data yang tengah dengan menggunakan rumus: (posisi awal + posisi akhir) / 2.
* Setelah itu data yang akan dicari dibandingkan dengan data yang berada di tengah, apakah data tersebut sama atau lebih kecil, atau lebih besar?
* Seandainya data tersebut  lebih besar, maka proses pencarian yang dicari dengan posisi awal adalah posisi tengah + 1
* Apabila data lebih kecil, maka proses pencarian yang dicari dengan posisi akhir adalah posisi tengah – 1
* Jika data sama dengan data yang di cari, berarti  data tersebut telah ketemu.

Kelebihan:

* Secara umum memiliki kompleksitas O(n log n).
* Algoritmanya sederhana dan mudah diterapkan pada berbagai bahasa pemrograman dan arsitektur mesin secara efisien.
* Dalam prakteknya adalah yang tercepat dari berbagai algoritma pengurutan dengan perbandingan, seperti merge sort dan heap sort.
* Melakukan proses langsung pada input (in-place) dengan sedikit tambahan memori.
* Bekerja dengan baik pada berbagai jenis input data (seperti angka dan karakter).

Kekurangan:

* Sedikit kesalahan dalam penulisan program membuatnya bekerja tidak beraturan (hasilnya tidak benar atau tidak pernah selesai).
* Memiliki ketergantungan terhadap data yang dimasukkan, yang dalam kasus terburuk memiliki kompleksitas O(n2).
* Secara umum bersifat tidak stable, yaitu mengubah urutan input dalam hasil akhirnya (dalam hal inputnya bernilai sama).
* Pada penerapan secara rekursif (memanggil dirinya sendiri) bila terjadi kasus terburuk dapat menghabiskan stack dan memacetkan program.

Codingan (dari kelompok 6):

#include <iostream>

using namespace std;

int main(){

int n,temp,find,low,high,mid,indeks;

cout<<"Binary Search"<<endl;

cout<<"============="<<endl;

cout<<"\nMasukan jumlah elemen yang ingin dimasukkan: ";

cin>>n;

int array[n];

cout<<"\nMasukan elemen-elemennya:"<<endl;

for(int i=0;i<n;i++){

cin>>array[i];

}

cout<<"\nSetelah diurutkan:"<<endl;

for (int i=0;i<n;i++){

for (int j=0;j<n;j++){

if(array[j]>array[j+1]){

temp=array[j];

array[j]=array[j+1];

array [j+1]=temp;

}

}

}

for (int i=0;i<n;i++){

cout<<array[i]<<" ";

}

cout<<"\n\nMasukan angka yang ingin dicari: "<<endl;

cin>>find;

low=0;

high=n-1;

while(low<=high){

mid=(low+high)/2;

if(array[mid]==find){

indeks=mid+1;

break;

}

else if(array[mid]>find){

high=mid-1;

}

else if(array[mid]<find){

low=mid+1;

}

}

cout<<endl<<find<<" berada di indeks ke-"<<indeks;

}

TERCEPAT DAN TERLAMBAT

**Sorting:**

Tercepat – Quick sort, karena data dipartisi menjadi kelompok kecil jadi tidak perlu waktu lama untuk saling membandingkan dan mengurutkan

Terlambat – Bubble sort, karena data yang ada satu persatu dibandingkan dan meskipun sudah urut masih tetap ikut dibandingkan sampai data terakhir

**Searching:**

Tercepat – Binary search, karena hanya butuh O(log N) kali pembandingan (sekitar 20 kali)

Terlambat – Linear search, karena butuh O(N) pembandingan (sekitar 500.000 kali)