Tugas 4

Praktikum Analisis Algoritma



Muhammad Naufal Monoarfa 140810160033

S-1 Teknik Informatika

Fakultas Matematika & Ilmu Pengetahuan Alam

Universitas Padjadjaran

1. Program
2. Bubble sort

#include <stdio.h>

#include <chrono>

#include <ctime>

#include <iostream>

using namespace std;

using namespace std::chrono;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

// A function to implement bubble sort

void bubbleSort(int arr[], int n)

{

int i, j;

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

// Last i elements are already in place

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

if (arr[j] > arr[j+1])

swap(&arr[j], &arr[j+1]);

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

}

// Driver program to test above functions

int main()

{

int arr[10000];

int n = sizeof(arr)/sizeof(arr[0]);

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

arr[i] = n - i;

}

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

bubbleSort(arr, n);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>( t2 - t1 ).count();

cout<<endl <<duration <<" microseconds" <<endl;

printf("Sorted array: \n");

printArray(arr, n);

}

1. Insertion sort

#include <stdio.h>

#include <chrono>

#include <ctime>

#include <iostream>

#include <math.h>

using namespace std;

using namespace std::chrono;

/\* Function to sort an array using insertion sort\*/

void insertionSort(int arr[], int n)

{

int i, key, j;

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

key = arr[i];

j = i - 1;

/\* Move elements of arr[0..i-1], that are

greater than key, to one position ahead

of their current position \*/

while (j >= 0 && arr[j] > key) {

arr[j + 1] = arr[j];

j = j - 1;

}

arr[j + 1] = key;

}

}

// A utility function to print an array of size n

void printArray(int arr[], int n)

{

int i;

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

printf("%d ", arr[i]);

printf("\n");

}

/\* Driver program to test insertion sort \*/

main()

{

int arr[1000];

int n = sizeof(arr)/sizeof(arr[0]);

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

arr[i] = n - i;

}

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

insertionSort(arr, n);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>( t2 - t1 ).count();

cout<<endl <<duration <<" microseconds" <<endl;

printf("\nSorted array is \n");

printArray(arr, n);

return 0;

}

1. Merge sort

#include <stdio.h>

#include <chrono>

#include <ctime>

#include <iostream>

using namespace std;

using namespace std::chrono;

// Merges two subarrays of arr[].

// First subarray is arr[l..m]

// Second subarray is arr[m+1..r]

void merge(int arr[], int l, int m, int r)

{

int i, j, k;

int n1 = m - l + 1;

int n2 = r - m;

/\* create temp arrays \*/

int L[n1], R[n2];

/\* Copy data to temp arrays L[] and R[] \*/

for (i = 0; i < n1; i++)

L[i] = arr[l + i];

for (j = 0; j < n2; j++)

R[j] = arr[m + 1+ j];

/\* Merge the temp arrays back into arr[l..r]\*/

i = 0; // Initial index of first subarray

j = 0; // Initial index of second subarray

k = l; // Initial index of merged subarray

while (i < n1 && j < n2)

{

if (L[i] <= R[j])

{

arr[k] = L[i];

i++;

}

else

{

arr[k] = R[j];

j++;

}

k++;

}

/\* Copy the remaining elements of L[], if there

are any \*/

while (i < n1)

{

arr[k] = L[i];

i++;

k++;

}

/\* Copy the remaining elements of R[], if there

are any \*/

while (j < n2)

{

arr[k] = R[j];

j++;

k++;

}

}

/\* l is for left index and r is right index of the

sub-array of arr to be sorted \*/

void mergeSort(int arr[], int l, int r)

{

if (l < r)

{

// Same as (l+r)/2, but avoids overflow for

// large l and h

int m = l+(r-l)/2;

// Sort first and second halves

mergeSort(arr, l, m);

mergeSort(arr, m+1, r);

merge(arr, l, m, r);

}

}

/\* UTILITY FUNCTIONS \*/

/\* Function to print an array \*/

void printArray(int A[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", A[i]);

printf("\n");

}

/\* Driver program to test above functions \*/

int main()

{

int arr[5000];

int n = sizeof(arr)/sizeof(arr[0]);

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

arr[i] = n - i;

}

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

mergeSort(arr, 0, n - 1);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>( t2 - t1 ).count();

cout<<endl <<duration <<" microseconds" <<endl;

printf("\nSorted array is \n");

printArray(arr, n);

return 0;

}

1. Selection sort

#include <stdio.h>

#include <chrono>

#include <ctime>

#include <iostream>

using namespace std;

using namespace std::chrono;

void swap(int \*xp, int \*yp)

{

int temp = \*xp;

\*xp = \*yp;

\*yp = temp;

}

void selectionSort(int arr[], int n)

{

int i, j, min\_idx;

// One by one move boundary of unsorted subarray

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

{

// Find the minimum element in unsorted array

min\_idx = i;

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

if (arr[j] < arr[min\_idx])

min\_idx = j;

// Swap the found minimum element with the first element

swap(&arr[min\_idx], &arr[i]);

}

}

/\* Function to print an array \*/

void printArray(int arr[], int size)

{

int i;

for (i=0; i < size; i++)

printf("%d ", arr[i]);

printf("\n");

}

// Driver program to test above functions

int main()

{

int arr[10000];

int n = sizeof(arr)/sizeof(arr[0]);

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

arr[i] = n - i;

}

high\_resolution\_clock::time\_point t1 = high\_resolution\_clock::now();

selectionSort(arr, n);

high\_resolution\_clock::time\_point t2 = high\_resolution\_clock::now();

auto duration = duration\_cast<microseconds>( t2 - t1 ).count();

cout<<endl <<duration <<" microseconds" <<endl;

printf("Sorted array: \n");

printArray(arr, n);

return 0;

}

1. Hasil Pengujian algoritma
2. Analisis Kompleksitas

## Merge Sort

## Insertion Sort

## Selection Sort

## Bubble Sort