Esercitazione 1

Codice:

#include <iostream>

#include <algorithm>

using namespace std;

void bubblesort( int A[]){

int count=0;

for (int i=0; i<5;i++){

for(int j=i+1;j<5;j++){

if( A[j]<A[i]){

swap(A[j],A[i]);

count++;

cout<<"count: "<<count<<endl;

}

}

}

}

void print( int A[]){

for(int z=0;z<5;z++){

cout<<A[z]<<endl;

}

}

int main(){

int A [5];

int B [5];

A[0]=9;

A[1]=1;

A[2]=0;

A[3]=5;

A[4]=4;

B[0]=3;

B[1]=1;

B[2]=2;

B[3]=3;

B[4]=0;

bubblesort(A);

print(A);

cout<<"Starting bubble sort on B"<<endl;

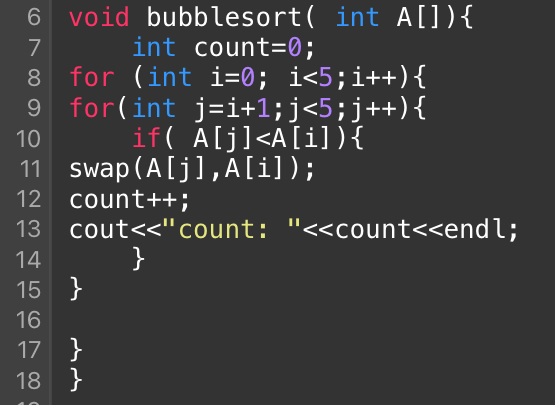
bubblesort(B);

print(B);

return 0;

}

Spiegazione: Per effettuare l’ordinamento previsto dalla traccia ho impiegato la strategia del bubble-sort, il quale ordina gli elementi effettuando dei paragoni e degli swap basati su due elementi adiacenti. La funzione implementata da me prende in input un Array di dimensione 5 e sfrutta un doppio for per girare l’array e per effettuare il paragone e lo scambio tra i numeri presenti nella struttura dati. In particolare il primo for serve a cambiare la key del nostro algoritmo ovvero l’elemento che prendo come riferimento per i paragoni e gli swap che effettuo nel secondo for, il quale ha lo scopo di analizzare tramite un if l’elemento key con i successivi ed effettuare nel caso in cui la condizione (A[j]<key) risulti vera uno swap tra gli elementi.

Analisi della Complessità: Per effettuare lo studio della complessità partiamo dalla sezione di codice di nostro interesse:



La complessità di questo algoritmo è n^2, poiché abbiamo un doppio for innestato e l’algoritmo non è ricorsivo il primo esegue n volte, mentre il secondo n-1, nell’analisi della complessità ho escluso lo swap poiché è una funzione presa dalla libreria <algorithm> anche se volendo affidarsi alle doc reperite online la funzione swap ha complessità O(n) nel caso peggiore( swap di due array completi) e nel caso medio swap di singoli elementi la sua complessità è O(1). Quindi nel nostro caso ha una complessità costante. La funzione bubble sort nell’ambito della complessità soffre di diverse pecche sia nel caso medio che nel caso pessimo che ottimo ha una complessità O(n^2) questo perché in tutti i casi deve eseguire obbligatoriamente entrambi i for per verificare che la condizione di ordinamento sia rispettata. Volendo essere pignoli ed analizzando tutte le funzioni impiegate nel main è stata implementata anche una print che ha complessità O(n) in tutti i casi di esecuzione poiché essa impiega un for per stampare tutti gli elementi dell’array dopo l’esecuzione dell’algoritmo di ordinamento. Una possibile risoluzione al problema per abbassare la complessità di esecuzione potrebbe essere il merge sort che come sappiamo è un agoritmo ricorsivo con complessità O(nlogn) quindi volendolo paragonare al bubble sort è un algoritmo efficientissimo. L’idea di base sarebbe la seguente ovvero decomporre l’array fino alla sua lunghezza minima:

