Scrapping Online Courses

CS-261 Mid Term Project



Project Supervisor

Mr. Samyan Qayyum Wahla

Project Members (ID:11)

|  |  |
| --- | --- |
| Member 1 Name: Muhammad umer Amjad | Member 1 Registration No.  2020-CS-61 |
| Member 2 Name: Salahuddin Sohail | Member 2 Registration No.  2020-CS-85 |

Department of Computer Science

University of Engineering and Technology, Lahore

Pakistan

Table of Contents

Selection Sort **1**

Merge Sort2

Quick Sort3

**Type chapter title (level 1)4**

Type chapter title (level 2)5

Type chapter title (level 3)6

# Selection Sort:

Selection sort is an algorithm based on linear search. It works iteratively and compare all the elements of the array one by one. As it is obvious from the description that it compares all the elements so it uses two iterative for loops. First loop holds the one value and the second is used to compare it iteratively with all the other elements. In this way we find the smallest value and swap it with the element on which base we were comparing. This process is literally time consuming because for a large array we have to do a lot of comparisons.

**Pseudo Code (Selection Sort):**

For i=1 to length of array

min = i

For j =i+1 to length of array

If array[min] > array[j]

min = j

swap (array[i], array[min])

**Python Code (Selection sort):**

from array import \*

def SelectionSort(arr):

for i in range(0,len(arr)):

minimum = i

for j in range(i+1,len(arr)):

if arr[minimum] > arr[j]:

minimum = j

rep = arr[i]

arr[i] = arr[minimum]

arr[minimum] = rep

size = int(input("enter size of array "))

arr = array('i' , [])

for k in range(0,size):

arr.append(int(input("enter elements of array ")))

SelectionSort(arr)

print("sorted array is ")

print(str(arr))

**Time Complexity Analysis:**

For i=1 to length of array O(n+1)

min = i O(n)

For j =i+1 to length of array O(n)\*O(n-(i+1))

If array[min] > array[j] O(n)\*O((n-1) - (i+1))

min = j O(n)\*O((n-1) - (i+1))

swap (array[i], array[min]) O(n)

**T(n) =** O(n+1) + O(n)+ O(n)\*O(n-(i+1))+ O(n)\*O((n-1) - (i+1))+ O(n)\*O((n-1) - (i+1))+ O(n)

**T(n) =** O(n^2)

Hence it will take bigo of n square time due to two iterative for loops.

**Proof of Correctness:**

Hence this algorithm is true because it first of all finds the smallest number from the array and then replace it with the first number. And then it again find the smallest number from j = i+1 so it will always give the true value.

**Three Strengths:**

The three strengths of Selection Sort are as follows

* It works for negative integers as well
* It will never make challenges no matter how complex is array
* It can also work for alphabets

**Three Weaknesses:**

The three weaknesses are as follows

* It will run in n square times
* It has to do comparisons
* It will depend on hardware

**Dry Run:**

enter size of array 8

enter elements of array 23

enter elements of array 6

enter elements of array 09

enter elements of array 122

enter elements of array -9

enter elements of array 4

enter elements of array 3

enter elements of array 98

sorted array is [-9, 3, 4, 6, 9, 23, 98, 122]

**Merge Sort:**

Merge Sort is a recursive algorithm. It works recursively and divide the array into smaller arrays until it reaches its base case. After reaching base case it then merge all the arrays together until arrays are sorted. The mergence of array works on comparison base it check first element of one array and compares it with the element of second array and place the smallest element first in the parent array. It consumes less time than selection sort because it divides the big problem into smaller problem.

**Pseudo Code (Merge Sort):**

mergeSort (arr,low,high)

if low < high

medium = low+high / 2

mergeSort (arr,low,medium)

mergeSort (arr,medium+1,high)

mergeArrays (arr,low,medium,high)

mergeArrays (arr,low,medium,high)

arr1 = 0 to medium

arr1[] = arr[x]

arr2 = medium+1 to high

arr1[] = arr[y]

while i < length of arr1 and j < length of arr2

if arr1[i] < arr2[j]

arr[k] = arr1[i]

else

arr[k] = arr[j]

**Python Code (Merge Sort):**

from array import \*

def mergeSort(arr ,low ,high):

if low < high:

medium = int((low+high)/2)

mergeSort(arr,low,medium)

mergeSort(arr,medium+1,high)

mergeArrays(arr,low,medium,high)

def mergeArrays(arr,low,medium,high):

size1 = (medium - low) + 1

size2 = high - medium

arr1 = array('i',[])

arr2 = array('i',[])

i =0

j =0

k =low

for x in range(0,size1):

arr1.append(arr[low+x])

for y in range(0,size2):

arr2.append(arr[(medium+1)+y])

while( i < len(arr1) and j < len(arr2)):

if (arr1[i] < arr2[j]):

arr[k] = arr1[i]

i =i+1

k =K+1

else:

arr[k] = arr2[j]

j =j+1

k =k+1

while(i<len(arr1)):

arr[k] = arr1[i]

i =i+1

k =k+1

while(j<len(arr2)):

arr[k] = arr2[j]

j =j+1

k =k+1

size3 = int(input("enter size of array "))

arr = array('i',[])

for m in range(0,size3):

arr.append(int(input("enter elements of arrqay ")))

low = 0

high = size3-1

mergeSort(arr,low,high)

print("sorted array is ")

print(str(arr))

**Time Complexity Analysis:**

def mergeSort(arr ,low ,high): O(1)

if low < high:

medium = int((low+high)/2) O(1)

mergeSort(arr,low,medium) O(log n)

mergeSort(arr,medium+1,high) O(log n)

mergeArrays(arr,low,medium,high) O(n)

hence the total sum of time will be

T(n) **= O(1) + O(log n ) + O(n)**

T(n) **= O(n**log**n)**

**Proof of Correctness:**

If we analyze the code correctly we will come to know that it divides the array from the middle and repeats the process until starting and ending point of the array becomes same. Hence after reaching base case it will merge them all together which is right so from the code we analyze that it will work properly even after each iteration hence it is correct.

**Three Strengths:**

* It takes less time
* It works recursively so we don’t need to repeat the code
* It is best for arrays of bigger size

**Three Weaknesses:**

* We must always take care of starting and ending index
* Not good for small inputs
* We can do more better than this

**Dry Run:**

enter size of array 5

enter elements of array 5

enter elements of array 4

enter elements of array 3

enter elements of array 2

enter elements of array 1

sorted array is [1, 2, 3, 4, 5]

**Quick Sort:**

Quick sort is also sorting algorithm that takes the same time as merge sort or insertion sort. It works on the principle of dividing the array in to two sub arrays and then apply sorting on both sides. First of all we assign of pivot from the array according to that pivot the algorithm divides the array into two halves. One half contains that elements that are smaller than the pivot and the other side have the greater element. Similarly we then assign a new pivot for side and for right side as well in this way array is sorted.

**Pseudo Code (Quick Sort):**

QuickSort(arr,low,high)

If low < high

Pivot = findPivot(arr, low ,high)

QickSort(arr,low,pivot-1)

QuickSort(arr,pivot+1,high)

findPivot(arr, low,high)

i =low

pivot = high

for j = low to legth of arr

if arr[j] < arr[pivot]

swap (arr[i] , arr[j])

swap (arr[i] ,arr[pivot])

**Python Code (Quick Sort):**

from array import \*

def QuickSort(arr,low,high):

if low < high:

pivot = findpivot(arr,low,high)

QuickSort(arr,low,pivot-1)

QuickSort(arr,pivot+1,high)

def findpivot(arr,low,high):

i =low

pivot = high

for j in range(low,high):

if arr[j] < arr[pivot]:

c = arr[i]

arr[i] = arr[j]

arr[j] = c

i =i+1

d = arr[i]

arr[i] = arr[pivot]

arr[pivot] = d

return i

size = int(input("enter size of array "))

arr = array('i',[])

for i in range(0,size):

arr.append(int(input("enter elements of array ")))

low =0

high = size -1

QuickSort(arr,low,high)

print(arr)

**Time Complexity Analysis:**

QuickSort(arr,low,high)

If low < high

Pivot = findPivot(arr, low ,high) O(log n)

QickSort(arr,low,pivot-1) O(log n)

QuickSort(arr,pivot+1,high) O(log n)

findPivot(arr, low,high)

i =low O(1)

pivot = high O(1)

for j = low to legth of arr O(n +1)

if arr[j] < arr[pivot] O(n)

swap (arr[i] , arr[j]) O(n)

swap (arr[i] ,arr[pivot]) O(1)

**T(n) =** O(log n ) + O(n+1) + O(n)+O(1)

**T(n) = O(n**log**n)**

**Proof of Correctness:**

The algorithm is also a iterative one it works correctly even if we divide the array into smaller sub arrays. So this algorithm holds true in every condition no matter before or after iteration.

**Three Strengths:**

* Takes less time than usual algorithms
* Divides the problem into smaller problems
* Recursive algorithm

**Three Weaknesses:**

* We can perform better than this
* Need comparison

**Dry Run:**

enter size of array 5

enter elements of array 5

enter elements of array 32

enter elements of array 4

enter elements of array 2

enter elements of array 89

[2, 4, 5, 32, 89]