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

Radix Sort5

Insertion Sort7

Count Sort8

Bubble Sort9

# Selection Sort:

It is also a technique to sort array either in ascending or descending array.it holds two subarrays at a time one is sorted and another is unsorted and by finding minimum it places that number in the beginning of sorted array. In this sorting every element from unsorted subarray is picked and move to sorted subarray. 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 an 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]

**Radix Sort:**

Radix Sort is one of the fastest algorithms. It is so fast because it uses count sort as backing. It does not do any comparisons so it will not take much time. Radix sort works in three phases first of all it finds the maximum number from the array and then finds number of places in that big integer. Then according to that places it adds zero on the left sides of the other indexes to make them equal to maximum number. In phase one it places the integers in the buckets according to their value at one’s place. Then to their ten’s or so on places. Rest of the concept can be seen is pseudo code.

**Pseudo Code (Radix Sort):**

findMax(arr)

max = findMax(arr)

for i=1 to length of array

add zero to the left of smaller numbers

according to one’s position of integers store them in the buckets ranges from 0 to 9

update the array by adding previous array with the next array

according to one’s place store the integer in the new array

repeat this process three times

**Python Code (Radix Sort):**

from array import \*

def findMax(arr):

maximum = 0

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

if (arr[maximum] < arr[i]):

maximum = i

return arr[maximum]

arr = []

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

for y in range(0,size):

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

arr1=[]

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

arr1.append(str(arr[k]))

maximum = findMax(arr)

for j in range(0,len(arr1)):

while(len(arr1[j]) < 3):

arr1[j] = "0" + arr1[j]

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Phase 1 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

array0 = []

for l in range(0,10):

array0.append(0)

for m in range(0,len(arr1)):

val = arr1[m]

value = int(val[2])

array0[value] = array0[value] + 1

for n in range(1,len(array0)):

array0[n] = array0[n] + array0[n-1]

One = len(arr1)-1

b1 =[]

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

b1.append("0")

while(One>=0):

val = arr1[One]

value = int(val[2])

array0[value] = array0[value] - 1

b1[array0[value]] = arr1[One]

One = One -1

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Phase 2 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

array1 = []

for p in range(0,10):

array1.append(0)

for q in range(0,len(b1)):

val = b1[q]

value = int(val[1])

array1[value] = array1[value] + 1

for r in range(1,len(array1)):

array1[r] = array1[r] + array1[r-1]

Two = len(b1)-1

b2 =[]

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

b2.append("0")

while(Two>=0):

val = b1[Two]

value = int(val[1])

array1[value] = array1[value] - 1

b2[array1[value]] = b1[Two]

Two = Two -1

# \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Phase 3 \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

array2 = []

for t in range(0,10):

array2.append(0)

for u in range(0,len(b2)):

val = b2[u]

value = int(val[0])

array2[value] = array2[value] + 1

for v in range(1,len(array2)):

array2[v] = array2[v] + array2[v-1]

Three = len(b2)-1

b3 =[]

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

b3.append("0")

while(Three>=0):

val = b2[Three]

value = int(val[0])

array2[value] = array2[value] - 1

b3[array2[value]] = b2[Three]

Three = Three -1

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

arr[x] = int(b3[x])

print("sorted array is ")

print(str(arr))

**Time Complexity Analysis:**

The time complexity of radix sort isO(d(n+b))\* time. Here b is the base if it is integer than base is 10 if alphabets then base is 26.

**Proof of Correctness:**

This algorithm also holds true for all cases.

**Three Strengths:**

* Don’t need Comparisons
* Uses Counting Sort

**Three Weaknesses:**

* Takes a little bit of more time
* Cannot Sort negative integers
* Lengthy Algorithm

**Dry Run:**

enter size of array 5

enter elements of array 6

enter elements of array 33

enter elements of array 6

enter elements of array 11

enter elements of array 0

sorted array is

[0, 6, 6, 11, 33]

**Insertion Sort:**

Insertion sort is basically a sorting technique which often use to sort array with respect to its position. By my mind it outperforms the sorting of an array in this way that an unsorted element compare itself with a sorted element(key) and then place itself in sorting manner by swapping itself with the key and the key also keeps on changing for every element of array. And we engage the insertion problem of an element in an array with this technique of sorting.

**Pseudo Code (Insertion Sort):**

Function of insertionsort(array):

for i to (1,len(array)):

j=i-1

key=array[i]

while(array[j]>key and j>=0):

array[j+1]=array[j]

j=j-1

array[j+1]=key

to get size=int(input("Enter size of array: "))

array=[]

for i to range(0,size):

take input in array.append(int(input("Enter elements of array: ")))

here we call function to get output

insertionsort(array)

print(array)

**Python Code (Insertion Sort):**

#Here is the python code of insertion sort

from array import\*

def insertionsort(array):

for i in range(1,len(array)):

j=i-1

key=array[i]

while(array[j]>key and j>=0):

array[j+1]=array[j]

j=j-1

array[j+1]=key

size=int(input("Enter size of array: "))

array=[]

for i in range(0,size):

array.append(int(input("Enter elements of array: ")))

insertionsort(array)

print(array)

**Time Complexity Analysis:**

The time complexity of insertion sort is O(nlogn)

Analysis is as follows.

=n

=n-1

=n-1

=(n-1) log(n)

=log(n)-1

=log(n)-1

=log(n)-1

**Proof of Correctness:**

Insertion Sort is also an iterative algorithm which uses only one for loop. It’s time complexity is also true for all inputs.

**Three Strengths:**

* Good for large number of inputs
* Uses only one iterative loop
* Closed to common sorting

**Dry Run:**

enter size of array 5

enter elements of array 6

enter elements of array 33

enter elements of array 6

enter elements of array 11

enter elements of array 0

sorted array is

[0, 6, 6, 11, 33]

**Count Sort:**

Count sort is sorting technique used whenever we want to sort uniformly distributed data and it plays a pivotal rule because it does not capitulate even sorting alphabets as well. And it is also based on keys between a specific range. And its code of conduct is that it outperforms sorting by arithmetic rules to calculate exact position of number in array means by decrementing and vice versa.

**Pseudo Code (Count Sort):**

Function of finding maximum(array):

maxi=array[0]

for j in range(1,len(array)):

if(array[j]>maxi):

maxi=array[j]

return maxi;

def appendvalue(array,array1):

for m to(0,len(array)):

array1[array[m]]= array1[array[m]]+1

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

array=[]

for i in range(0,size):

take input in array (int(input("Enter elements of array")))

maxi=maximum(array)

array1=[]

for k in range(0,maxi+1):

array1.append(0)

calling a function appendvalue(array,array1)

array2=[]

for n in range(0,len(array1)):

while(array1[n]!=0):

array1[n]=array1[n]-1

array2.append(n)

print(array2) to get output

**Python Code (Count Sort):**

from array import\*

def maximum(array):

maxi=array[0]

for j in range(1,len(array)):

if(array[j]>maxi):

maxi=array[j]

return maxi;

def appendvalue(array,array1):

for m in range(0,len(array)):

array1[array[m]]= array1[array[m]]+1

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

array=[]

for i in range(0,size):

array.append(int(input("Enter elements of array")))

maxi=maximum(array)

array1=[]

for k in range(0,maxi+1):

array1.append(0)

appendvalue(array,array1)

array2=[]

for n in range(0,len(array1)):

while(array1[n]!=0):

array1[n]=array1[n]-1

array2.append(n)

print(array2)

**Time Complexity Analysis:**

Time Complexity of Count Sort is O(n+k).

**Three Strengths:**

* One of the quickest algorithms
* Don’t need comparison
* Good for alphabets sorting

**Three Weaknesses:**

* Not good for negative integers
* If difference between integers is very large we have to make a large length of array

**Dry Run:**

enter size of array 5

enter elements of array 6

enter elements of array 33

enter elements of array 6

enter elements of array 11

enter elements of array 0

sorted array is

[0, 6, 6, 11, 33]

**Bubble Sort:**

It is also a technique to sort array either in ascending or descending array.it holds two subarrays at a time one is sorted and another is unsorted and by finding minimum it places that number in the beginning of sorted array. In this sorting every element from unsorted subarray is picked and move to sorted subarray.

**Pseudo Code (Bubble Sort):**

For i=1 to length of array

For j=1 to length of array-1

If arr[j] < arr[j+1]

Swap(arr[i] , arr[j+1])

**Python Code (Bubble Sort):**

from array import \*

def BubbleSort(arr):

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

for j in range(0,len(arr)-1):

if arr[j] > arr[j+1]:

c = arr[j]

arr[j] = arr[j+1]

arr[j+1] = c

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

arr = array('i',[])

for k in range(0,size):

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

BubbleSort(arr)

print(arr)

**Time Complexity Analysis:**

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

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

If arr[j] < arr[j+1]

Swap(arr[i] , arr[j+1])

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

**Dry Run:**

enter size of array 5

enter elements of array 3

enter elements of array 4

enter elements of array 66

enter elements of array 1

enter elements of array 3

[1, 3, 3, 4, 66]