Index

|  |  |  |  |
| --- | --- | --- | --- |
| S.No | Exercise | Date | Page no |
| 1 | Write a Linear search Algorithum and Program. | 6-09-2021 | 2-3 |
| 2 | Write Bubble sort and selection sort Algorithum and Program | 13-09-2021 | 4-5 |
| 3 | Write Min Max Algorithum and Program. | 20-09-2021 | 6-8 |
| 4 | Write Merge sort and Quick sort Algorithum and program. | 27-09-2021 | 9-11 |
| 5 | Implement Linear and Binary Search determine the time complexity to search a element.Repeat the experiment for different values of n the number of elements in the list to be searched. | 4-10-2021 | 12-14 |
| 6 | Design an algorithum to find min and max from the given array with the time complexity not more than 3n/2. | 11-10-2021 | 15-16 |

Exercise-1

Write a Linear search Algorithum and Program.

Algorithum

step1: Start from the leftmost element of given arr[]

step2: Then one by one compare element x with each element of arr[]

step3: If x matches with any of the element, return the index value.

step4: If x doesn’t match with any of elements in arr[] then return -1 or element not found.

Code

first = []

size\_array = int(input("Enter size of list: \t"))

for n in range(size\_array):

num = int(input("Enter any number: \t"))

first.append((num))

x = int(input("\nEnter number to search: \t"))

found = False

for i in range(len(first)):

if first[i] == x:

found = True

print("\n%d found at position %d" % (x, i))

break

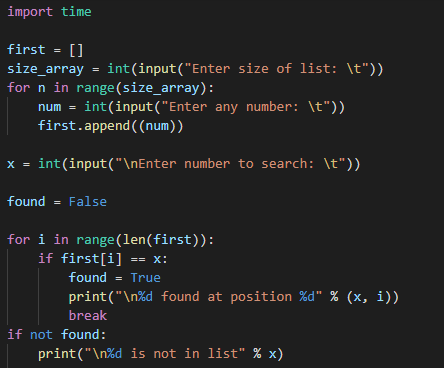
if not found:

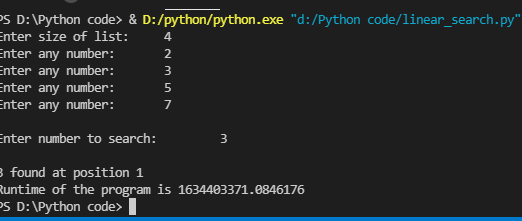
print("\n%d is not in list" % x)

time.sleep(1)

end = time.time()

print(f"Runtime of the program is {end }")





Space Complexity-O(1)

Time Complexity-O(n)

Excersise-2

Write Bubble sort and selection sort Algorithum and Program.

Algorithum-Bubble sort

step 1:First take all elements in a list

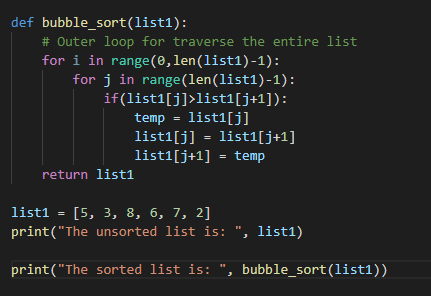
step 2:Then compare the first number with second number in list list[i] > list[i+1]

step 3:Swap the number if the list[i+1] is greater than list[i]

step 4:Repeat step3 until list gets arranged in ascending order and entire list is sorted.

Time complexity-o(n)

Space complexity-o(1)





Algorithum-Selection sort

Step 1 - Select the first element of the list.

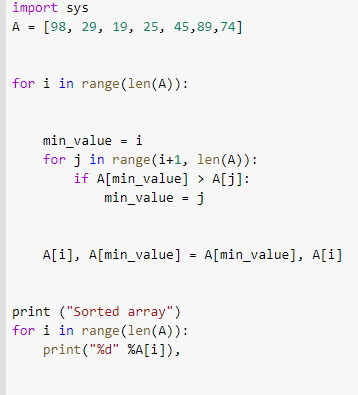
Step 2: Compare the selected element with all the other elements in the list.

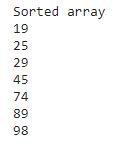
Step 3: In every comparision, if any element is found smaller than the selected element (for Ascending order), then both are swapped.

Step 4: Repeat the same procedure with element in the next position in the list till the entire list is sorted.

Time complexity-o(n\*n)

space complexity-o(1)





Excersise-3

Write Min Max Algorithum and Program.

Algorithum-Min Max

Algorithum(min,max)

Step 1:maximum\_num=arry[i]

Step 2:minimun\_num=arry[i]

step 3:for in range i = 1 to n

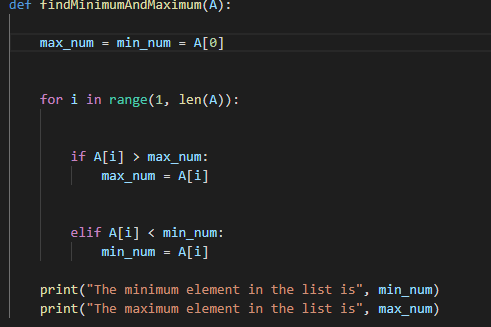
step 4:arry[i] > max then max=arry[i]

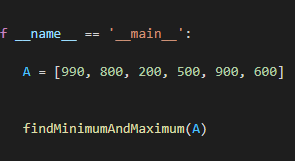
step 5:if arry[i] < min then min=arry[i]

step 6:repeat until array gets sorted with min and max value

Step 7:return(maximum\_number,minimum\_number)

Time complexity-o(n)







Algorithum-Insertion sort

Step 1:If the element is the first element that is already sorted

Step2:then take the next element and store it in a key

Step3:now compare the key with other elements in array.

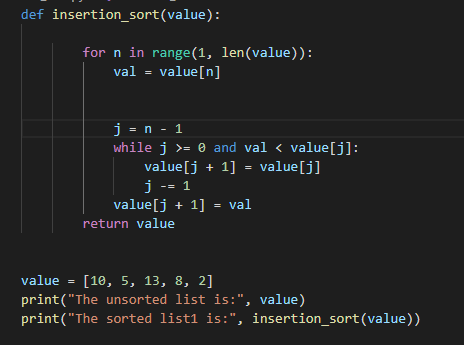
Step4:if the element is the sorted array is smaller than the current element,then move to the next element.

Step5:then shift the greater elements in array towards right

Step6:then insert the value

Step7:repeat until the array is sorted

Time complexity-o(n2)





Exercise-4

Write Merge sort and Quick sort Algorithum and program.

Algorithum-Merge Sort

Algorithum Mergesort(l,h)

step 1:First take lower value in array and highest value in array.

step 2:if(l < h)

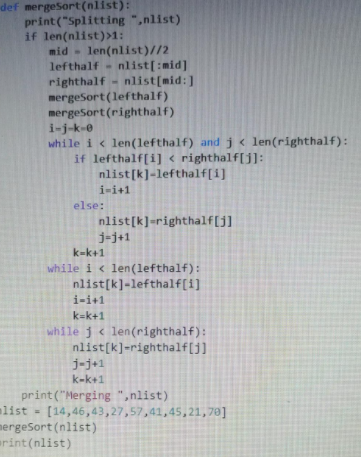
step 3:mid=(l+h)/2

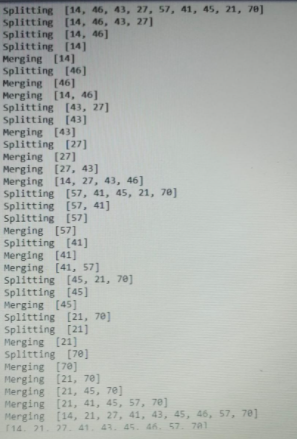
step 4:mergesort(l,mid) taken lower value and mid value from array.

step 5: mergesort(mid+1,h)

step 6:mergesort(l,mid,h)

Time complexity of MergeSort is O(n\*Log n)





Algorithum-Quick sort

Algorithum QuickSort(largestnumber , highest number)

step 1:first in array check the element from right side with the starting number

step 2:then right side number is smaller than starting number than interchange both the number.

step 3: then again check from left side the largest number compare it with other end if number is smaller than left side number than interchange with that number.

step 4:If from left side in an array the element get sorted then element is pivot because left number of element get sorted in ascending number.

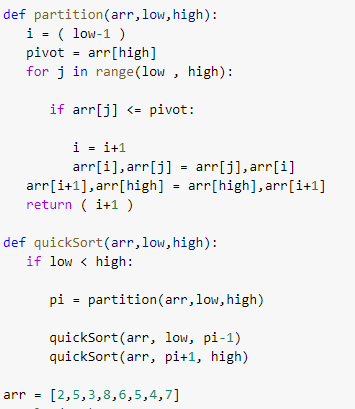
step 5:after pivot then divide it into two parts.

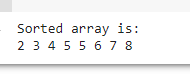
step 6:first part is sorted that is aranged.

step 7:Second part after pivot is need to sort.

step 8:Repeat step 2 and 3 until elements gets sorted an arranged in asecending order.

complexity - O(n logn)





Exercise-5

Implement Linear and Binary Search determine the time complexity to search a element.Repeat the experiment for different values of n the number of elements in the list to be searched.

Algorithum - Linear search

step1: Start from the leftmost element of given arr[]

step2: Then one by one compare element x with each element of arr[]

step3: If x matches with any of the element, return the index value.

step4: If x doesn’t match with any of elements in arr[] then return -1 or element not found.

Code

first = []

size\_array = int(input("Enter size of list: \t"))

for n in range(size\_array):

num = int(input("Enter any number: \t"))

first.append((num))

x = int(input("\nEnter number to search: \t"))

found = False

for i in range(len(first)):

if first[i] == x:

found = True

print("\n%d found at position %d" % (x, i))

break

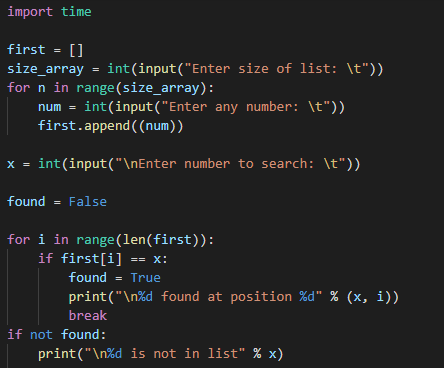
if not found:

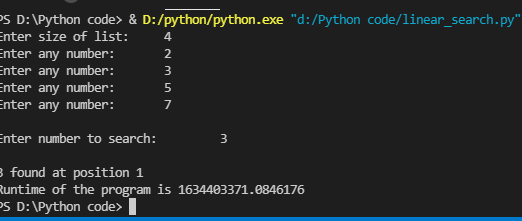
print("\n%d is not in list" % x)

time.sleep(1)

end = time.time()

print(f"Runtime of the program is {end }")



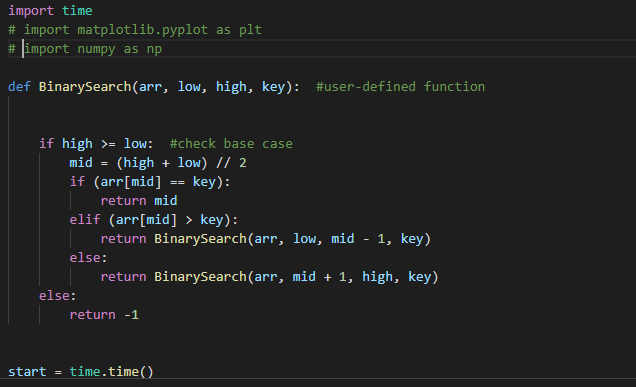


Space Complexity-O(1)

Time Complexity-O(n)

Binary Search

1. Compare x with the middle element.
2. If x matches with the middle element, we return the mid index.
3. Else If x is greater than the mid element, then x can only lie in the right half subarray after the mid element. So we recur for the right half.
4. Else (x is smaller) recur for the left half.





Exercise-6

Design an algorithum to find min and max from the given array with the time complexity not more than 3n/2.

Algorithum(array)

step 1:array[i]

step 2:enter the number of element in array

step 3:i % 2==0 number will be even

step 3:check the number the whether number is odd or even.

step 4:when number is even execute whole program and if the number of element is odd then program will not execute.

step 5:now the number of element is even so the array will divide into half.

step 6:then from subproblem1 the array[i] > max then max=array[i]

step 7:then repeat for min value also array[i] < min then min=array[i]

step 8:same as for subproblem 2.

step 9:repeat until array get sorted with min and max value

