Design & Analysis Of Algorithm

Lab Experiment -1

NAME : SURIYAPRAKASH.C

ROLL NO:CH.EN.U4CSE20170

SUBJECT: DESIGN & ANALYSIS OF ALGORITHM

SUBJECT CODE: 19CSE302

Submitted to – Mrs. Ashwini,

Department of CSE,

ASE Chennai campus.

EX NO: 1

1. **Sorting Techniques**
    a. **Insertion sort**
    b. **Bubble sort**
    c. **Selection sort**
    d. **Merge sort**
    e. **Quick sort**
    f. **Heap sort**
    g. **Bucket sort**

a) Insertion sort

AIM:

To write an algorithm to implement insertion sort.

ALGORITHM:

To sort an array of size N in ascending order:

1. Iterate from arr[1] to arr[N] over the array.
2. Compare the current element (key) to its predecessor.
3. If the key element is smaller than its predecessor, compare it to the elements before. Move the greater elements one position up to make space for the swapped element.

CODE SCREEN:

```
def insertionSort(array):

    for step in range(1, len(array)):
        key = array[step]
        j = step - 1
        while j >= 0 and key < array[j]:
            array[j + 1] = array[j]
```

```
            j = j - 1
        array[j + 1] = key

data = [9, 5, 1, 4, 3]
insertionSort(data)
print('Sorted Array in Ascending Order:')
print(data)
```

OUTPUT SCREEN :

```
PS D:\python> & C:/Users/HP/AppData/Local/Programs/Python/Python310/python.exe d:/python/DAA/insertion.py
Sorted Array in Ascending Order:
[1, 3, 4, 5, 9]
PS D:\python>
```

TIME COMPLEXITY:

Worst case – O(n^2)

Best case – O(n)

RESULT:

I have studied and understood the insertion sort in python language and executed the program successfully.

b)Bubble Sort:

AIM:

To write an algorithm to implement bubble sort .

ALGORITHM:

begin BubbleSort(list)

for all elements of list

if list[i] > list[i+1]

swap(list[i], list[i+1])

end if

end for

return list

end BubbleSort

CODE SCREEN:

```
def bubble_sort(list1):
    for i in range(0,len(list1)-1):
        for j in range(len(list1)-1):
            if(list1[j]>list1[j+1]):
                temp = list1[j]
                list1[j] = list1[j+1]
                list1[j+1] = temp
    return list1

list1 = [4,8,2,1,5,3]
print("The unsorted list is: ", list1)
print("The sorted list is: ", bubble_sort(list1))
```

OUTPUT SCREEN :

```
PS D:\python> & C:/Users/HP/AppData/Local/Programs/Python/Python310/python.exe d:/python/DAA/bubble.py
The unsorted list is:  [4, 8, 2, 1, 5, 3]
The sorted list is:  [1, 2, 3, 4, 5, 8]
PS D:\python>
```

TIME COMPLEXITY:

Worst case – O(n^2)

Best case – O(n)

RESULT:

I have studied and understood the bubble sort in python language and executed the program successfully.

c)Selection sort

AIM:

To write an algorithm to implement selection sort .

ALGORITHM:

1) Set Min to location 0 in Step 1.
2) Look for the smallest element on the list.
3) Replace the value at location Min with a different value.
4) Increase Min to point to the next element
5) Continue until the list is sorted.

CODE SCREEN:

```
def selectionSort( itemsList ):
    n = len( itemsList )
    for i in range( n - 1 ):
        minValueIndex = i
        for j in range( i + 1, n ):
            if itemsList[j] < itemsList[minValueIndex] :
                minValueIndex = j
        if minValueIndex != i :
            temp = itemsList[i]
            itemsList[i] = itemsList[minValueIndex]
            itemsList[minValueIndex] = temp
    return itemsList

el = [45,87,12,32,9,60,11,5]
print("The unsorted list is: ", el)
print("The sorted list is: ",selectionSort(el))
```

OUTPUT SCREEN :

```
PS D:\python> & C:/Users/HP/AppData/Local/Programs/Python/Python310/python.exe d:/python/DAA/selection.py
The unsorted list is:  [45, 87, 12, 32, 9, 60, 11, 5]
The sorted list is:  [5, 9, 11, 12, 32, 45, 60, 87]
PS D:\python>
```

TIME COMPLEXITY:

Worst case – O(n^2)

Best case – O(n^2)

RESULT:

I have studied and understood the selection sort in python language and executed the program successfully.

d) Merge Sort

AIM:

To write an algorithm to implement Merge sort .

ALGORITHM:

step 1: start

step 2: declare array and left, right, mid variable

step 3: perform merge function.

if left > right

return

mid= (left+right)/2

mergesort(array, left, mid)

mergesort(array, mid+1, right)

merge(array, left, mid, right)

step 4: Stop

CODE SCREEN:

```
def mergeSort(array):
    if len(array) > 1:
        r = len(array)//2
        L = array[:r]
```

```
        M = array[r:]

        mergeSort(L)
        mergeSort(M)

        i = j = k = 0
        while i < len(L) and j < len(M):
            if L[i] < M[j]:
                array[k] = L[i]
                i += 1
            else:
                array[k] = M[j]
                j += 1
            k += 1
        while i < len(L):
            array[k] = L[i]
            i += 1
            k += 1

        while j < len(M):
            array[k] = M[j]
            j += 1
            k += 1

def printList(array):
    for i in range(len(array)):
        print(array[i], end=" ")
    print()
if __name__ == '__main__':
    array = [12,98,70,2,65,32]

    print("Initial array is: ",array)

    mergeSort(array)

    print("Sorted array is: ")
    printList(array)
```

OUTPUT SCREEN :

```
PS D:\python> & C:/Users/HP/AppData/Local/Programs/Python/Python310/python.exe d:/python/DAA/merge.py
Initial array is:  [12, 98, 70, 2, 65, 32]
Sorted array is:
2 12 32 65 70 98
PS D:\python>
```

TIME COMPLEXITY:

Best case – O(n logn)

Worst case – O(n logn)

RESULT:

I have studied and understood the merge sort in python language and executed the program successfully.

e) Quick sort

AIM:

To write an algorithm to implement quick sort.

ALGORITHM:

```
quickSort(array, leftmostIndex, rightmostIndex)
  if (leftmostIndex < rightmostIndex)
    pivotIndex <- partition(array,leftmostIndex, rightmostIndex)
    quickSort(array, leftmostIndex, pivotIndex - 1)
    quickSort(array, pivotIndex, rightmostIndex)
partition(array, leftmostIndex, rightmostIndex)
  set rightmostIndex as pivotIndex
  storeIndex <- leftmostIndex - 1
  for i <- leftmostIndex + 1 to rightmostIndex
  if element[i] < pivotElement
    swap element[i] and element[storeIndex]
    storeIndex++
  swap pivotElement and element[storeIndex+1]
return storeIndex + 1
```

CODE SCREEN:

```python
def partition(array, low, high):
    pivot = array[high]
    i = low - 1
    for j in range(low, high):
        if array[j] <= pivot:
            i = i + 1
            (array[i], array[j]) = (array[j], array[i])
    (array[i + 1], array[high]) = (array[high], array[i + 1])
    return i + 1

def quickSort(array, low, high):
    if low < high:
        pi = partition(array, low, high)
        quickSort(array, low, pi - 1)
        quickSort(array, pi + 1, high)

data = [1, 7, 4, 1, 10, 9, -2]
print("Unsorted Array")
print(data)

size = len(data)

quickSort(data, 0, size - 1)

print('Sorted Array in Ascending Order:')
print(data)
```

OUTPUT SCREEN :

```
PS D:\python> & C:/Users/HP/AppData/Local/Programs/Python/Python310/python.exe d:/python/DAA/quick.py
Unsorted Array
[1, 7, 4, 1, 10, 9, -2]
Sorted Array in Ascending Order:
[-2, 1, 1, 4, 7, 9, 10]
PS D:\python>
```

TIME COMPLEXITY:

Worst case – O(n^2)

Best case – O(n logn)

RESULT:

I have studied and understood the quick sort in python language and executed the program successfully.

f) Heap Sort:

AIM:

To write an algorithm to implement heap sort .

ALGORITHM:

heapify(array)

Root = array[0]

Largest = largest(array[0] , array [2 * 0 + 1]/ array[2 * 0 + 2])

if(Root != Largest)

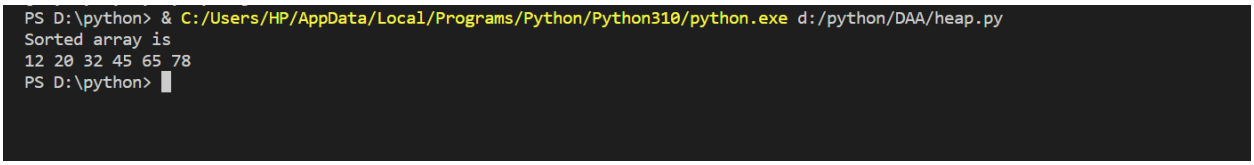
Swap(Root, Largest)

CODE SCREEN:

```
def heapify(arr, n, i):
      largest = i
      l = 2 * i + 1
      r = 2 * i + 2

      if l < n and arr[i] < arr[l]:
          largest = l

      if r < n and arr[largest] < arr[r]:
          largest = r

      if largest != i:
          arr[i], arr[largest] = arr[largest], arr[i]
          heapify(arr, n, largest)


def heapSort(arr):
```

```
      n = len(arr)
      for i in range(n//2, -1, -1):
          heapify(arr, n, i)

      for i in range(n-1, 0, -1):
          arr[i], arr[0] = arr[0], arr[i]
          heapify(arr, i, 0)


arr = [65,45,78,12,32,20]
heapSort(arr)
n = len(arr)
print("Sorted array is")
for i in range(n):
     print("%d " % arr[i], end='')
```

OUTPUT SCREEN :



TIME COMPLEXITY:

Best case – O(n logn)

Worst case – O(n logn)

RESULT:

I have studied and understood the heap sort in python language and executed the program successfully.

g) Bucket Sort:

AIM:

To write an algorithm to implement bucket sort .

ALGORITHM:

Bucket Sort(A[])

1. Let B[0....n-1] be a new array
2. n=length[A]
3. for i=0 to n-1
4. make B[i] an empty list
5. for i=1 to n
6. do insert A[i] into list B[n a[i]]
7. for i=0 to n-1
8. do sort list B[i] with insertion-sort
9. Concatenate lists B[0], B[1],....., B[n-1] together in order

End

CODE SCREEN:

```
def bucketSort(array):
    bucket = []
    for i in range(len(array)):
        bucket.append([])
    for j in array:
        index_b = int(10 * j)
        bucket[index_b].append(j)

    for i in range(len(array)):
        bucket[i] = sorted(bucket[i])

    k = 0
    for i in range(len(array)):
        for j in range(len(bucket[i])):
            array[k] = bucket[i][j]
            k += 1
    return array

array = [.47, .51 , .42, .32, .33, .52, .37, .23]
print("Sorted Array in descending order is")
```

```
print(bucketSort(array))
```

OUTPUT SCREEN :

```
PS D:\python> & C:/Users/HP/AppData/Local/Programs/Python/Python310/python.exe d:/python/DAA/bucket.py
Sorted Array in descending order is
[0.23, 0.32, 0.33, 0.37, 0.42, 0.47, 0.51, 0.52]
PS D:\python>
```

TIME COMPLEXITY:

Worst case – O(n^2)

Best case – O(n)

RESULT:

I have studied and understood the Bucket sort in python language and executed the program successfully.

THANK YOU !!