CS261-Data Structure and Algorithms

Mid Project Algorithms Design(Fall 2021)

**Project Details**

| Group Number | *G5* |
| --- | --- |
| Registration Number of Group Members | 2020-CS-155(Qamar Mehar)  2020-CS-108(Muzamil Iqbal) |
|  |  |

|  |  |
| --- | --- |
| **Project Description:** | **Our Project names as hotel pricing Analytics , in our project we will make the an online system which is used to reserve a room in the hotel and it will show which rooms are available and which are reserved. Also show the location of the hotel ratings and address of the hotel.**  **In our project we will scrap the data for our project from different sites including**   1. **Booking .com** 2. **TripAdvisor.com** 3. **Hotel.com**   **etc.**  **Each website has large amount of information about the hotels all over the world.**  **We will use these following entries to scrap like**   1. **Hotel Pricing per room** 2. **Hotel Name** 3. **Room Type** 4. **Hotel Address** 5. **Hotel Rating** 6. **Hotel Reviews** 7. **Hotel Number of Bed rooms**   **And also much more as soon as possible according to the needs of project**  **We will do the code of our project Hotel Pricing Analytics that describes the functionalities of sorting the data which we will scrap from the different sites and sort it using different king of algorithms like selection sort , Merge Sort etc. We will do the best efforts in developing this project in a well and different manner using the pyqt5 and Qt designer** |
|  |  |

|  |
| --- |
| **UI Design** |
| **Login Page:** |
| **Home Page:** |
| **Scrapping Page:** |
| **About Project Page:** |

|  |  |
| --- | --- |
| **Motivation Of Project** | **Every Person who travel outside from the home have tension in searching hotel.**  **So , We design a online system help the people to book any hotel**  **Also , In the world passenger face lot of difficulties to choose the best hotel for his accommodation. Many people does not know how face these difficulties. So in this situations we designing this the project who will help the passengers to choose the best hotel for his accommodation. Also provide large of information of hotel such what is the Name of Hotel ,Price of Hotel and etc?**  **Our Project only not for passenger but also for all kind of people who face difficulties to choose the best hotel for his accommodation.**  **It is good project not for only high class people but also middle class people.** |

|  |  |
| --- | --- |
| **Audience:** |  |

|  |  |
| --- | --- |
| **Business Need:** | **Our Project is useful for any kind of information about the hotel such as hotel name , hotel price , hotel address etc. Every Person who travel outside from the home have tension in searching hotel.**  **So , We design a online system help the people to book any hotel**  **Also , In the world passenger face lot of difficulties to choose the best hotel for his accommodation. Many people does not know how face these difficulties. So in this situations we designing this the project who will help the passengers to choose the best hotel for his accommodation. Also provide large of information of hotel such what is the Name of Hotel ,Price of Hotel and etc?**  **Our Project only not for passenger but also for all kind of people who face difficulties to choose the best hotel for his accommodation.**  **It is good project not for only high class people but also middle class people.** |

|  |  |
| --- | --- |
| **Scrapping:** | **Scrapping is very useful tool for assemble large amount of data for any kind of project. We have scrapped almost (400000)**  **Only one website which is known as booking.com**  **A large amount of data is available but the shortage of time we scrape only (400000) data which is necessary of our project** |

|  |  |
| --- | --- |
| **Collaboration:** | **We have a good collaboration during developing this project. We have done almost work together. During developing this project we have face lot of difficulties but we solve these problems together. It is good effort both of us to develop this project. Also me and my partner motivate each other. Also me and my partner help each other.** |

|  |  |
| --- | --- |
| **Task Division:** | **We have done almost work together.**  **2020-cs-108 Scrap all data from different website**  **2020-cs-108 Design Algorithms**  **2020-cs-108 and 155 Design Project Proposal together**  **2020-cs-155 Design UI Interface**  **Integration done both of us together** |

|  |
| --- |
| **Algorithms Design:** |

|  |
| --- |
| **Selection Sort** |

|  |  |
| --- | --- |
| Description of Algorithms | We have a sets of card and we want to arrange these cards in increasing or decreasing order of rank . One simple thing we can do initially  We keep all the cards in our left hands and then first we can select the minimum card out of these cards and move it to the right hand. Now , once again from we can select the minimum and moving to the right hand ,next previous card in the right. We can go on repeating the processes until the finally right hand will be sorted arrangement of cards in case of decreasing order |
| Pseudo code of Algorithms | A[]=[8,1,5,7,6,9]  for i=0 to n:  min=i  for j =i+1 to n:  if(A[j]<A[i]):  min=j  if(minIndex!==i):  swap the list Elements |
| Code in Python | #Selection Sort  A=[7,5,2,1,8]  def SelectionSort(A):  for i in range(len(A)):  min = i  for j in range(i+1, len(A)):  if A[min] > A[j]:  min = j    #Swape the Array in Selection Sort  temp=A[i]  A[i]=A[min]  A[min]=temp  #Driver code of python  SelectionSort(A)  print("The Sorted Array is:")  for i in range(len(A)):  print(A[i]) |
| Time Complexity | A[]=[8,1,5,7,6,9]  for i=0 to n: c1\*(n)  min=i c2\*(n-1)  n  for j =i+1 to n: Σ(n-i+1)  i=0  n  if(A[j]<A[i]): Σ(n-i)  i=0  n  min=j Σ(n-i)  i=0  if(minIndex!==i): n-1  swape the list Elements n-1  Total Time=c1+c2+c3+c4+c5  n  T(t)=c1\*(n)+c2\*(n-1)+c3\* Σ(n-i+1)+  n n i=0  c4\* Σ(n-j)+c5\* Σ(n-j)+c6\*(n-1)+c7\*(n-1)  i=0 i=0  T(t)=n+n-1+n^2-n(n-1)/2+n+ n^2-n(n-1)/2++ n^2-n(n-1)/2+n-1+n-1  By Solving it we get  T(t)=O(n^2)  Best Case:  O(n^2)  Worst Case:  O(n^2) |
| Proof of Correctness | By using loop invariant we show the proof of correctness first of all we check the condition  If(A(min)>A[j]) if the condition is true then the min element in sorted form then again check the condition until A[1,2,3,,,,,n] the loop is terminates |
| Tree Strengths | 1. Selection sort sometimes is more efficient than more algorithms it just depends on the condition 2. It is simplest of form of sort algorithms that manage easily 3. It works simply rather than others |
| Three Weakness | 1. Selection sort is not stable because every time we compare the elements than time complexity is more 2. Swape the elements in array again and again 3. for large number of inputs it is not more efficients |
| Dry run on small input | A=[7,5,1,2,8]  for i=0 to n:  min=n  for j=i+1 to n:  if(A[j]<A[min])  min=j  //swap Numbers     |  |  |  | | --- | --- | --- | | Array  A=[7,5,1,2,8]  A=[1,5,7,2,8]  A=[1,2,7,5,8]  A=[1,2,5,7,8]  A=[1,2,5,7,8] | Condition  If(5<7)  If(7<1)  If(5<2)  If(7<5)  If(8<7)  The Sorted Array is:  A=[1,2,5,7,8] | Swap  No Swapping  Temp=5  A[min]=7  A[j]=1  Swap  Swap  No swap | |

|  |
| --- |
| **Insertion Sort** |

|  |  |
| --- | --- |
| Description of Algorithms | Insertion Sort Similar to those cards who are in our hands . First of all We pick the card and insert into according to the level of cards means if we pick 10 than we shift the ten 10 it position. In Insertion sort nothing compare we just shift the values . In Insertion sort we do not swape values we just shift the values . In sertion sort first of all we set the key value than  Shift this key value in sorted position. |
| Pseudo code of Algorithms | A[]=[8,1,5,7,6,9]  for j=2 to A.lenth  key=A[j]  i=j-1  while(i>0 and A[j]<key):  A[i+1]=A[i]  I=i-1  A[i+1]=key |
| Code in Python | #Insertion Sort  A=[5, 7, -8, 9, 10, 4, -7, 0,-12, 1, 6, 2, 3, -4, -15, 12]  #Declarization of Array  def InsertionSort(A):  for i in range(1,len(A)):  key=A[i]  print(key)  j=i-1  while(j>=0 and A[j]>key):  A[j+1]=A[j]  j=j-1  A[j+1]=key  #Drive Code  print("The Given Array is:")  for i in range(len(A)):  print(A[i])  InsertionSort(A)  print("The Sorted Array is:")  for i in range(len(A)):  print(A[i]) |
| Time Complexity | A[]=[8,1,5,7,6,9]  for j=2 to A.length: c1\*(n)  key=A[j] c2\*(n-1)  i=j-1 c3\*(n-2)  n  while i>0 and A[j]>key: c4\*Σt(j)  j=2  n  if(A[i+1]=A[i]): c5\*Σ(j-1)  j=2  n  i=i-1 c6\* Σ(j-1)  j=2  A[i+1]=key c7\*(n-1)  Total Time=c1+c2+c3+c4+c5+c6+c7    T(t)=c1\*(n)+c2\*(n-1)+c3\* (n-2)+  n n n  c4\* Σ(j)+c5\* Σ(j-1)+c6 Σ(j-1)+c7\*(n-1)  j=2 j=2 j=2  By Solving it we get  T(t)=O(n^2)  Best Case: (When Array is sorted)  O(n)  Worst Case:(When Array is unsorted)  O(n^2) |
| Proof of Correctness | By using loop invariant we show the proof of correctness first of all we take some key =A[i] than compare it with values of by using loop  While(j>0 and A[j]>key) if the condition is true then the min element in sorted form then again check the condition until A[1,2,3,,,,,n] the loop is terminates |
| Tree Strengths | 1. Insertion sort is more efficient for small 2. It is simplest of form of sort algorithms that manage easily 3. It works simply rather than others |
| Three Weakness | 1. Selection sort is not stable because for large number of inputs it does not work correctly 2. Shift the elements in array again and again 3. for large number of inputs it is not more efficients |
| Dry run on small input | A=[5,2,-8,1,-2]  for i=1 to n  key=A[i]  j=i-1  while(A[j]>key and j>0)  A[j+1]=A[j]  j=j-1  A[j+1]=key   |  |  |  | | --- | --- | --- | | Array  A=[5,2,-8,1,-2]  A=[2,5,-8,1,-2]  A=[2,-8,5,1,-2]  A=[-8,-2,1,5,2]  The Sorted Array is  A=[-8,-2,1,5,2] | Condition  While(0>0 and 5>2)  While(1>0 and 2>-8)  While(2>0 and -8>1) | Shifting  A[j+1]=5  Key=2  Shift  No shift | |

|  |  |
| --- | --- |
| **Bubble Sort** | |
| Description of Algorithms | Bubble sort is almost similar to selection sort but the difference is in selection sort we find the min element than swape but in Bubble sort we just compare the element to next element than swape it . This procedure gives us a complete sorted algorithm. But the time complexity of both algorithms is almost same . |
| Pseudo code of Algorithms | A[]=[8,1,5,7,6,9]  for i=0 to n:  for j =i+1 to n:  if(A[j]<A[i]):  #swap the elements  if(minIndex!==i):  swape the list Elements |
| Code in Python | #Bubble Sort  #Declarization of function  A=[6, 0, 8, 2, 3, 0, 4, 0, 1]  def BubbleSort(A):  for i in range(len(A)):  for j in range(i+1, len(A)):  if A[i] > A[j]:  temp=A[i]  A[i]=A[j]  A[j]=temp    #Drive Code  print("The given Array is:")  for i in range(len(A)):  print(A[i])  BubbleSort(A)  print("The Sorted Array is:")  for i in range(len(A)):  print(A[i]) |
| Time Complexity | A[]=[8,1,5,7,6,9]  for i=0 to n: c1\*(n)  n  for j =i+1 to n: Σ(n-i+1)  i=0  n  if(A[j]<A[i]): Σ(n-i)  i=0  swape the list Elements n-1  Total Time=c1+c2+c3+c4+c5  n n  T(t)=c1\*(n)+c2\* Σ(n-i+1)+ c3\*Σ(n-i)+c4\*(n-1)  n i=0 i=0  T(t)=n+n^2-n(n-1)/2+n+ n^2-n(n-1)/2+n-1  By Solving it we get  T(t)=O(n^2)  Best Case: (When Array is sorted)  O(n)  Worst Case:(When Array is unsorted)  O(n^2) |
| Proof of Correctness | By using loop invariant we show the proof of correctness first of all we check the condition  If(A(i)>A[j]) if the condition is true then we swape element in sorted form then again check the condition until A[1,2,3,,,,,n] the loop is terminates |
| Tree Strengths | 1. Bubble sort sometimes is more efficient than more algorithms it just depends on the condition 2. It is simplest of form of sort algorithms that manage easily 3. It works simply rather than others |
| Three Weakness | 1. Bubble sort is not stable because every time we compare the elements than time complexity is more 2. Compare the elements in array again and again 3. for large number of inputs it is not more efficients |
| Dry run on small input | A=[5,2,7,6,1]  for i=0 to n  for j=i+1 to n:  if(A[j]>A[i])  temp=a[i]  A[i]=A[j]  A[j]=temp   |  |  |  | | --- | --- | --- | | Array  A=[5,2,7,6,1]  A=[2,5,7,6,1]  A=[2,5,7,6,1]  A=[2,5,7,6,1]  A=[1,5,7,6,2]  A=[1,5,7,6,2]  A=[1,5,7,6,2]  A=[1,2,7,6,5]  The Sorted Array is:  A=[1,2,7,6,5] | Condition  If(5>2)  If(2>5)  If(2>7)  If(2>6)  If(2>1)  If(5>7)  If(5>6)  If(5>2) | Swapping  Swap(True)  False  False  False  True(swap)  False  False  True(swap) | |
| **Merge Sort** | |
| Description of Algorithms | Merge Sort is also called Divide and coquer method . In merge sort we have a List of different elements . We divide this list into sub-lists and than  Call it recursively and combine the list again. We Divide list in sub-lists until the base condition is come when the base condition is come than we combine the list again and call the function recursively. In this Way we get the Sorted list of elements. |
| Pseudo code of Algorithms | MergeSort(Array,left,right)  #Midle element  Middle=left+(right-1)/2  MergeSort(Array,left,Middle)  MergeSort(Array,Middle,right)  Merge(Array,left,Middle,right) |
| Code in Python | #Merge Sort      def Merge(arr, l, m, r):  n1 = m - l + 1  n2 = r - m    #Now We are Creating the two empyt list  L = [0] \* (n1)  R = [0] \* (n2)    # Copy the data in list  for i in range(0, n1):  L[i] = arr[l + i]    for j in range(0, n2):  R[j] = arr[m + 1 + j]    #declarization of three variables  i = 0  j = 0  k = l    while i < n1 and j < n2:  if L[i] <= R[j]:  arr[k] = L[i]#Copy left array in initial array  i += 1  else:  arr[k] = R[j]  j += 1  k += 1    # Copy the remaining elements of L[], if there    while i < n1:  arr[k] = L[i]  i += 1  k += 1    # Copy the remaining elements of R[], if there  # are any  while j < n2:  arr[k] = R[j]  j += 1  k += 1      def mergeSort(arr, l, r):  if l < r:      m = l+(r-l)//2    #first half of the array  mergeSort(arr, l, m)  #second half of the array  mergeSort(arr, m+1, r)  Merge(arr, l, m, r)      # Driver code to test above  arr = [4, 1, 3, 9, 7]  n = len(arr)  print("The Given Array is:")  for i in range(n):  print(arr[i]),    mergeSort(arr, 0, n-1)  print("The Sorted Array Is:")  for i in range(n):  print(arr[i]), |
| Time Complexity | As we know merge sort is divide and conquer method so its time complexity is given as:    T(n)=T(n/2)+T(n/2)+n  For base case always  T(1)=1  Put n=2  T(2)=T(2/2)+T(2/2)+2  T(2)=4  Put n=4  T(4)=T(4/2)+T(4/2)+4  T(4)=12  Put n=8  T(8)=T(8/2)+T(8/2)+8  T(8)=30  .  T(n)=log n+1  Now we take log of input values  Log1=0+1=1  Log2=1+1=2  Log3=2+1=3  Log n=log n+1  Now  T(n)/n=log n+1  T(n)=n log n+1  Hence the Time Complexity of Merge sort is  O(n log n)  Best Case:(When array is sorted)  O(n log n)  Worst Case:(When array is unsorted)  O(n log n) |
| Proof of Correctness | By using the induction hypothesis we proof the correctness of merge sort algorithm  Step 1:  T(n)=n log n+n  T(1)=1  Put n=1  T(1)=1log1+1  Log1=0  1=1  Hence T(1) is true  Step 1=2:  Assume that T(k) is also true  T(k)=K log K + K (1)  Step 3:  We want to prove that T(k+1) is true  T(k+1)=(K+1)log(K+1)+K+1  T(k+1)k=(K+1)log(K+1)+K+1  T(k^2)+T(K)=(K+1)log(K+1)+K+1  From (1)  T(K^2)+k log K+K= K+1)log(K+1)+K+1  So it is clear that T(k+1) is also true  Hence merge is correct logically |
| Tree Strengths | 1. Merge sort is most efficient algorithm has running time is very good 2. Merge sort is better than the selection insertion and bubble sort   Because it time complexity is better than the other   1. It works simply rather than others |
| Three Weakness | 1. Merge sort takes an extra space to store the divided parts of array means to store the sub-Arrays 2. It works slowly for small number of arrays 3. Even if the Array is sorted than it repeats the whole processes of Cursive definitions |
| Dry run on small input | A=[2,5,1,4,7]  2 5 1 4 7  2 5 1 4 7  2 5 1 4 7  2 5 4 7  2 5  Condition check  1 2 5  1 2 4 5 7  Hence the sorted Array is:  A=[1,2,4,5,7] |

|  |  |
| --- | --- |
| **Quick Sort** | |
| Description of Algorithms | Quick Sort is a divide and conquer algorithm. Quick sort first divides a large array into smaller sub-Arrays the low elements and the high elements. Quick Sort can then recursively sort the sub-arrays.  Pick an element as pivot from the list or array. Reorder the array so that all elements with values less than the pivot come before the pivot . While all elements with values greater than the pivot come after it. After this partitioning the pivot is in its it final position.  We choice pivot any elements |
| Pseudo code of Algorithms | A[]=[8,1,5,7,6,9]  QuickSort(A, low ,high)  If(low<high)  pivot=PartitionFun(A,low , high)  //Elements before pivots  QuickSort(A,low,pivot-1)  //Elements after pivots  QuickSort(A,pivot+1,high)  Partition(A,low,high)  Pivot=A[high]  i=low-1  for j=low to high-1  if(A[i]<pivot)  i++  //swap the elements of A[i] and A[j]  //Swap A[i+1] and A[j]  Return(i+1) |
| Code in Python | #Quick Sort  arr= [4,1,3,9,7]  def quickSort(arr, low, high):  if (low < high):  pi = partition(arr, low, high)  quickSort(arr, low, pi - 1)  quickSort(arr, pi + 1, high)  def partition (arr, low, high):  pivot = arr[high]  i = (low - 1)  for j in range(low,high):  if (arr[j] < pivot):  i=i+1  temp=arr[i]  arr[i]=arr[j]  arr[j]=temp    temp=arr[i+1]  arr[i+1]=arr[high]  arr[high]=temp  return (i + 1)  #Driver Code  quickSort(arr, 0, len(arr)-1)  partition (arr, 0, len(arr)-1)  print(arr) |
| Time Complexity | A[]=[8,1,5,7,6,9]  The Time complexity of Quick sort is:  Best Case:  The time complexity of best case of Quick Sort we find in Merge sort given behind:    T(n)=n log (n)  Average Case:    The time complexity of Average case of Quick Sort we find in Merge sort given behind:    T(n)=n log (n)  Worst Case:  But in worst case it increase the time complexity  Which is given it:  T(n)=O(n^2) |
| Proof of Correctness | By using the Induction hypothesis and loop variant we show that the correctness of Quick Sort By Using Induction  By using the induction hypothesis we proof the correctness of merge sort algorithm  Step 1:  T(n)=n log n+n  T(1)=1  Put n=1  T(1)=1log1+1  Log1=0  1=1  Hence T(1) is true  Step 1=2:  Assume that T(k) is also true  T(k)=K log K + K (1)  Step 3:  We want to prove that T(k+1) is true  T(k+1)=(K+1)log(K+1)+K+1  T(k+1)k=(K+1)log(K+1)+K+1  T(k^2)+T(K)=(K+1)log(K+1)+K+1  From (1)  T(K^2)+k log K+K= K+1)log(K+1)+K+1  So it is clear that T(k+1) is also true  Hence merge is correct logically |
| Tree Strengths | 1. Quick sort is about two to three time faster than the merge sort and heap sort 2. The time complexity of Merge sort is good for best and average case 3. It works simply rather than others |
| Three Weakness | 1. Quick Sort is a Recursive algorithm but then if the recursion is not fond than it is more complicated 2. It takes very large or more time in its worse case 3. In worst case it is not sufficient |
| Dry run on small input | A=[4,1,3,9,7]    4 1 3 9 7(pivot)  Small than pivot large than pivot  4 1 3(pivot) 9  1 4  Hence the sorted Array is:  A=[1,3,4,7,9] |
| **K-Select Sort** | |
| Description of Algorithms | K-Select is a Selection Algorithm to find the kth smallest element in an unordered list . It is related to the quick sort algorithm  Quick Sort is a divide and conquer algorithm. Quick sort first divides a large array into smaller sub-Arrays the low elements and the high elements. Quick Sort can then recursively sort the sub-arrays.  Pick an element as pivot from the list or array. Reorder the array so that all elements with values less than the pivot come before the pivot . While all elements with values greater than the pivot come after it. After this partitioning the pivot is in its it final position.  We choice pivot any elements |
| Pseudo code of Algorithms | A[]=[8,1,5,7,6,9]  def quickSelect(A, left, right, k):  if left == right:  return arr  pivotIndex = partition(A, left, right)  if k == pivotIndex:  return arr[k]  elif k < pivotIndex:  right =pivotIndex - 1  return quickSelect(A, left, right, k)  else:  left =pivotIndex + 1  return quickSelect(A, left, right, k)Partition(A,low,high)  Pivot=A  i=low-1  for j=low to high-1  if(A<pivot)  i++  //swap the elements of A[i] and A[j]  //Swap A[i+1] and A[j]  Return(i+1) |
| Code in Python | #Result  arr= [1, 20, 14, 3, 22, 11]  def quickSelect(arr, left, right, k):  if left == right:  return arr[left]  pivotIndex=0  pivotIndex = partition(arr, left, right)  if k == pivotIndex:  return arr[k]  elif k < pivotIndex:  right =pivotIndex - 1  return quickSelect(arr, left, right, k)  else:  left =pivotIndex + 1  return quickSelect(arr, left, right, k)  def partition (arr, low, high):  pivot = arr[high]  i = (low - 1)  for j in range(low,high):  if (arr[j] < pivot):  i=i+1  temp=arr[i]  arr[i]=arr[j]  arr[j]=temp    temp=arr[i+1]  arr[i+1]=arr[high]  arr[high]=temp  return (i + 1)  #Driver Code  k=int(input("Enter the kth smallest element"))  quickSelect(arr, 0, len(arr)-1, k-1)  partition (arr, 0, len(arr)-1)  print(arr[k]) |
| Time Complexity | A[]=[8,1,5,7,6,9]  The Time complexity of Quick sort is:  Best Case:  The time complexity of best case of Quick Sort we find in Merge sort given behind:    T(n)=n log (n)  Average Case:    The time complexity of Average case of Quick Sort we find in Merge sort given behind:    T(n)=n log (n)  Worst Case:  But in worst case it increase the time complexity  Which is given it:  T(n)=O(n^2) |
| Proof of Correctness | By using the Induction hypothesis and loop variant we show that the correctness of Quick Sort By Using Induction  By using the induction hypothesis we proof the correctness of merge sort algorithm  Step 1:  T(n)=n log n+n  T(1)=1  Put n=1  T(1)=1log1+1  Log1=0  1=1  Hence T(1) is true  Step 1=2:  Assume that T(k) is also true  T(k)=K log K + K (1)  Step 3:  We want to prove that T(k+1) is true  T(k+1)=(K+1)log(K+1)+K+1  T(k+1)k=(K+1)log(K+1)+K+1  T(k^2)+T(K)=(K+1)log(K+1)+K+1  From (1)  T(K^2)+k log K+K= K+1)log(K+1)+K+1  So it is clear that T(k+1) is also true  Hence merge is correct logically |
| Tree Strengths | 1. Quick sort is about two to three time faster than the merge sort and heap sort 2. The time complexity of Merge sort is good for best and average case 3. It works simply rather than others |
| Three Weakness | 1. Quick Sort is a Recursive algorithm but then if the recursion is not fond than it is more complicated 2. It takes very large or more time in its worse case 3. In worst case it is not sufficient |
| Dry run on small input | A=[4,1,3,9,7]  The output is:  A=[1,3,4,7,9] |

|  |  |
| --- | --- |
| **Heap Sort** | |
| Description of Algorithms | Heap Sort is the comparison data structure which depends upon the binary heap. It is similar to the selection sort in heap sort first we find the minimum or maximum element as same like selection sort  When we find the min element than it is called the min-heap when we find the max element than is called the max-heap.  By Continuously repeating this processes we sort the array . In heap sort A heap is complete binary tree |
| Pseudo code of Algorithms | A[]=[8,1,5,7,6,9]  for i=0 to n:  for j =i+1 to n:  if(A[j]<A[i]):  #swap the elements  if(minIndex!==i):  swape the list Elements |
| Code in Python | def Heap(A, n, i):  Largest = i  l = 2 \* i + 1  r = 2 \* i + 2      if l < n and A[Largest] < A[l]:  Largest = l      if r < n and A[Largest] < A[r]:  Largest = r      if Largest != i:  #Swapping  temp=A[i]  A[i]=A[Largest]  A[Largest]=temp    Heap(A, n, Largest)  def HeapSort(A):  n = len(A)      for i in range(n//2 - 1, -1, -1):  Heap(A, n, i)      for i in range(n-1, 0, -1):  #Swapping  temp=A[i]  A[i]=A[0]  A[0]=temp  #Calling Function  Heap(A, i, 0)      # Driver code  A = [12, 11, 13, 5, 6, 7]  HeapSort(A)  n = len(A)  print("The sorted Array is:")  for i in range(n):  print(A[i]) |
| Time Complexity | A[]=[8,1,5,7,6,9]  for i=0 to n: c1\*(n)  n  for j =i+1 to n: Σ(n-i+1)  i=0  n  if(A[j]<A[i]): Σ(n-i)  i=0  swape the list Elements n-1  Total Time=c1+c2+c3+c4+c5  n n  T(t)=c1\*(n)+c2\* Σ(n-i+1)+ c3\*Σ(n-i)+c4\*(n-1)  n i=0 i=0  T(t)=n+n^2-n(n-1)/2+n+ n^2-n(n-1)/2+n-1  By Solving it we get  T(t)=O(n^2)  Best Case: (When Array is sorted)  O(n)  Worst Case:(When Array is unsorted)  O(n^2) |
| Proof of Correctness | By using loop invariant we show the proof of correctness first of all we check the condition  If(A(i)>A[j]) if the condition is true then we swap element in sorted form then again check the condition until A[1,2,3,,,,,n] the loop is terminates |
| Tree Strengths | 1. The Heap sort is good efficient but less efficient than the merge or quick sort. 2. Heap sort it is easy to understand 3. It works simply rather than others |
| Three Weakness | 1. The Big Disadvantage of heap sort is it takes lot of time and wastage a lot of memory 2. Heap sort is less efficient to merge and quick sort because it waste the large of time to sort Array 3. Heap sort is not stable algorithm as compare to quick sort |
| Dry run on small input | A=[5,2,7,6,1]  //array in heap  5  2 7  6 1  Hence the sorted Array is:  A=[1,2,5,6,7] |
| **Counting Sort** | |
| Description of Algorithms | Counting sort is not a comparison algorithm it just works on the keys elements. It works by counting the number of objects having different key values . Than doing some operation to calculate the position of each object in the output sequence. It is used to count the elements to sort the array Than the given result of Array is sorted. |
| Pseudo code of Algorithms | A[]=[8,1,5,7,6,9]  K=range of elements  Created two empty arrays  Count=k+1  Output=same as input array  for i=0 to length-1  j=key(input[i)  count[j]=count[j]+1  for i=1 to k  count[j]=count[j]-1  for i=length(input)-1 to 0  j=key[input[i])  output[count]=input[i] |
| Code in Python | # Counting Sort  def CountingSort(Input):  max\_element = int(max(Input))  min\_element = int(min(Input))  range\_of\_elements = max\_element - min\_element + 1  count =[]  for i in range(range\_of\_elements):  count.append(0)  output=[]  for i in range(len(Input)):  output.append(0)  for i in range(0, len(arr)):  count[Input[i]-min\_element] += 1  for i in range(1, len(count)):  count[i] += count[i-1]  for i in range(len(Input)-1, -1, -1):  output[count[Input[i] - min\_element] - 1] = Input[i]  count[arr[i] - min\_element] -= 1  for i in range(0, len(Input)):  Input[i] = output[i]  return Input  # Driver program to test above function  Input = [-5, -10, 0, -3, 8, 5, -1, 10]  answer = CountingSort(Input)  print("The Sorted Array are:")  print( str(answer))    # Driver code  A = [12, 11, 13, 5, 6, 7]  HeapSort(A)  n = len(A)  print("The sorted Array is:")  for i in range(n):  print(A[i]) |
| Time Complexity | A[]=[8,1,5,7,6,9]  for i=0 to n: c1\*(n)  n  for j =i+1 to n: Σ(n-i+1)  i=0  n  if(A[j]<A[i]): Σ(n-i)  i=0  swape the list Elements n-1  Total Time=c1+c2+c3+c4+c5  n n  T(t)=c1\*(n)+c2\* Σ(n-i+1)+ c3\*Σ(n-i)+c4\*(n-1)  n i=0 i=0  T(t)=n+n^2-n(n-1)/2+n+ n^2-n(n-1)/2+n-1  By Solving it we get  T(t)=O(n^2)  Best Case: (When Array is sorted)  O(n)  Worst Case:(When Array is unsorted)  O(n^2) |
| Proof of Correctness | By using loop invariant we show the proof of correctness first of all we check the condition  If(A(i)>A[j]) if the condition is true then we swap element in sorted form then again check the condition until A[1,2,3,,,,,n] the loop is terminates |
| Tree Strengths | 1. The main advantage of counting sort is its time complexity 2. The time complexity of the counting is very efficient than others 3. It works simply rather than others |
| Three Weakness | 1. The main disadvantage of counting sort it works only integer values   Not others.   1. It can also use for only arrays or list 2. Counting sort is not better than the merge sort |
| Dry run on small input | A=[2,1,0,1,0,2,3,8]  //find the Range of list  Range=max-min  K=Range  K=8   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 2 | 1 | 0 | 1 | 0 | 2 | 3 | 8 |   C[i]=0   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |   The Count Array is:   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 2 | 2 | 2 | 1 | 0 | 0 | 0 | 1 |   Sum of Previous and Next   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 2 | 4 | 6 | 7 | 7 | 7 | 7 | 8 |   The Output Array is:   |  |  |  |  |  |  |  |  | | --- | --- | --- | --- | --- | --- | --- | --- | | 0 | 0 | 1 | 1 | 2 | 2 | 3 | 8 |   This is final sorted Array by using counting sort |
| **Radix Sort** | |
| Description of Algorithms | Radix Sort is a sorting technique to used the digit from least significant digit to most significant digit  Counting sort is not a comparison algorithm it just works on the keys elements. It works by counting the number of objects having different key values . Than doing some operation to calculate the position of each object in the output sequence. It is used to count the elements to sort the array Than the given result of Array is sorted. |
| Pseudo code of Algorithms | K=range of elements  Created two empty arrays  Count=k+1  Output=same as input array  for i=0 to length-1  j=key(input[i)  count[j]=count[j]+1  for i=1 to k  count[j]=count[j]-1  for i=length(input)-1 to 0  j=key[input[i])  output[count]=input[i]  radixSort(arr):    # Find the maximum number to know number of digits  max1 = max(arr)  exp = 1  while max1 / exp > 0:  countingSort(arr, exp)  exp \*= 10 |
| Code in Python | def countingSort(arr, exp1):    n = len(arr)    # The output array elements that will have sorted arr  output = [0] \* (n)    # initialize count array as 0  count = [0] \* (10)    # Store count of occurrences in count[]  for i in range(0, n):  index = arr[i] // exp1  count[index % 10] += 1  for i in range(1, 10):  count[i] += count[i - 1]    # Build the output array  i = n - 1  while i >= 0:  index = arr[i] // exp1  output[count[index % 10] - 1] = arr[i]  count[index % 10] -= 1  i -= 1  i = 0  for i in range(0, len(arr)):  arr[i] = output[i]    def radixSort(arr):    # Find the maximum number to know number of digits  max1 = max(arr)  exp = 1  while max1 / exp > 0:  countingSort(arr, exp)  exp \*= 10      # Driver code  arr = [170, 45, 75, 90, 802, 24, 2, 66]    # Function Call  radixSort(arr)    for i in range(len(arr)):  print(arr[i]) |
| Time Complexity | A[]=[8,1,5,7,6,9]  for i=0 to n: c1\*(n)  n  while max1 / exp > 0:: Σ(n-i+1)  i=0  n  if(A[j]<A[i]): Σ(n-i)  i=0  Total Time=c1+c2+c3+c4+c5  n n  T(t)=c1\*(n)+c2\* Σ(n-i+1)+ c3\*Σ(n-i)+c4\*(n-1)  n i=0 i=0  T(t)=n+n^2-n(n-1)/2+n+ n^2-n(n-1)/2+n-1  By Solving it we get  T(t)=O(n^2)  Best Case:  The best case time complexity is  O(n)  Worst Case:  The worst case time complexity is:  O(n^2) |
| Proof of Correctness | By using loop invariant we show the proof of correctness first of all we check the condition  If(A(i)>A[j]) if the condition is true then we swap element in sorted form then again check the condition until A[1,2,3,,,,,n] the loop is terminates |
| Tree Strengths | 1. It is fast algorithm when the keys are sort and range of the elements is short 2. The main advantage of the radix sort is it is stable algorithm 3. It works simply rather than others |
| Three Weakness | 1. It is much less flexible algorithm 2. Heap sort is less efficient to merge and quick sort because it waste the large of time to sort Array 3. It takes more space as compared to others |
| Dry run on small input | Inputs+  A=[2,1,3,5,1,5,6]  Outpus:  A=[1,1,2,3,5,6] |
| **Bucket Sort** | |
| Description of Algorithms | Heap Sort is the comparison data structure which depends upon the binary heap. It is similar to the selection sort in heap sort first we find the minimum or maximum element as same like selection sort  When we find the min element than it is called the min-heap when we find the max element than is called the max-heap.  By Continuously repeating this processes we sort the array . In heap sort A heap is complete binary tree |
| Pseudo code of Algorithms | A[]=[8,1,5,7,6,9]  Create n empty buckets  Do following for every array element arr[i]  Insert the arr[i] into bucket n\*array[i]  Sort individual buckets using insertion sort  Concatenate all sorted buckets |
| Code in Python | def insertionSort(b):  for i in range(1, len(b)):  up = b[i]  j = i - 1  while j >= 0 and b[j] > up:  b[j + 1] = b[j]  j -= 1  b[j + 1] = up  return b    def bucketSort(x):  arr = []  slot\_num = 10  for i in range(slot\_num):  arr.append([])    for j in x:  index\_b = int(slot\_num \* j)  arr[index\_b].append(j)  for i in range(slot\_num):  arr[i] = insertionSort(arr[i])  k = 0  for i in range(slot\_num):  for j in range(len(arr[i])):  x[k] = arr[i][j]  k += 1  return x    # Driver Code  x = [0.897, 0.565, 0.656,  0.1234, 0.665, 0.3434]  print("Sorted Array is")  print(bucketSort(x)) |
| Time Complexity | A[]=[8,1,5,7,6,9]  for i=0 to n: c1\*(n)  n  while j >= 0 and b[j] > up: Σ(n-i+1)  i=0  n  if(A[j]<A[i]): Σ(n-i)  i=0  swape the list Elements n-1  Total Time=c1+c2+c3+c4+c5  n n  T(t)=c1\*(n)+c2\* Σ(n-i+1)+ c3\*Σ(n-i)+c4\*(n-1)  n i=0 i=0  T(t)=n+n^2-n(n-1)/2+n+ n^2-n(n-1)/2+n-1  By Solving it we get  T(t)=O(n^2)  Best Case:  The best case time complexity of bucket sort is  O(n)  Worst Case:  The Worst case time complexity of is O(n^2) |
| Proof of Correctness | By using loop invariant we show the proof of correctness first of all we check the condition  If(A(i)>A[j]) if the condition is true then we swap element in sorted form then again check the condition until A[1,2,3,,,,,n] the loop is terminates |
| Tree Strengths | 1. Bucket sort is more efficient than the bubble sort 2. The main advantages of bucket sort are it reduced the number of comparison. 3. Bucket sort is stable form of algorithm |
| Three Weakness | 1. The main disadvantage of the bucket sort is that it is not applying for the all the data types 2. bucket sort is less efficient to merge and quick sort because it waste the large of time to sort Array 3. Bucket sort is not applicable for large number of cases |
| Dry run on small input | Inputs:  A=[3,1,5,6,7,7,8]  Outputs:  A=[1,3,5,6,7,7,8] |

Final Application:

We have scrap almost 4 Billion data from different hotel websites. Also develop beautiful GUI as soon as possible. All the sorting technique are applied but some are missing due to the shortage of time. Also we have does not perform the multiple level sorting in this GUI.

(best of luck)