BCSE204P- Design and Analysis of Algorithms Lab
E1-Slot-L43-L44 Lab
In-Lab Practice (IPS) Exercise-2

Name: R. Muhmmad Abrar
Register no: 21BRS1713

Any Three question Output:

Q2).

2. Given n 2-dimensional points $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$, Design an algorithm that follows the 'Divide-Conquer-Combine' strategy to

   (a) arrange the $n - points$ in an increasing order of the x coordinates

   (b) arrange the $n - points$ in an increasing order of the y coordinates

   (c) arrange the $n-points$ in decreasing order of the value $(x-coordinate+ y - coordinate)/2)$ coordinates

   For each of the above algorithms, execute the algorithm with an appropriate code. Compute $t(n)$ for each of the algorithm.

Code:

merge-sort-2.cpp > printArray(int [], int)

```cpp
#include <iostream>
#include<ctime>
using namespace std;

void merge(int arr[], int p, int q, int r) {
  int n1 = q - p + 1;
  int n2 = r - q;
  int L[n1], M[n2];
  for (int i = 0; i < n1; i++)
    L[i] = arr[p + i];
  for (int j = 0; j < n2; j++)
    M[j] = arr[q + 1 + j];
  int i, j, k;
  i = 0;
  j = 0;
  k = p;
  while (i < n1 && j < n2) {
    if (L[i] <= M[j]) {
      arr[k] = L[i];
      i++;
    } else {
      arr[k] = M[j];
      j++;
    }
    k++;
  }
    while (i < n1) {
    arr[k] = L[i];
    i++;
    k++;
  }
  while (j < n2) {
    arr[k] = M[j];
    j++;
    k++;
  }
}
```

```cpp
void mergeSort(int arr[], int l, int r) {
  if (l < r) {
    int m = l + (r - l) / 2;

    mergeSort(arr, l, m);
    mergeSort(arr, m + 1, r);

    merge(arr, l, m, r);
  }
}

void printArray1(int arr[], int size) {
  for (int i = size-1; i > -1; i--)
    cout << arr[i] << " ";
  cout << endl;
}

void printArray(int arr[], int size) {
  for (int i = 0; i < size; i++)
    cout << arr[i] << " ";
  cout << endl;
}
int main() {
    clock_t tStart = clock();
    int n,value,j=0,k=0;
    cin>>n;
    int arr1[n],arr2[n],arr3[n];
    for (int i=0; i<n; i++){
        cin>>value;
        arr1[j]=value;
        cin>>value;
        arr2[j]=value;
        j++;
    }
    for (int i=0; i<n; i++){
        arr3[k] = (arr1[i] + arr2[i]) / 2;
        k++;
    }
    int size = n;
    mergeSort(arr1, 0, size - 1);
    double time1=(double)(clock() - tStart)/CLOCKS_PER_SEC;
    cout<<"Time taken is "<<time1<<endl;
    cout << "Sorted array: \n";
    printArray(arr1, size);
    mergeSort(arr2, 0, size - 1);
    double time2=(double)(clock() - tStart)/CLOCKS_PER_SEC;
    cout<<"Time taken is "<<time2<<endl;
    cout << "Sorted array: \n";
    printArray(arr2, size);
    mergeSort(arr3, 0, size - 1);
    double time3=(double)(clock() - tStart)/CLOCKS_PER_SEC;
    cout<<"Time taken is "<<time3<<endl;
    cout << "Sorted array: \n";
    printArray1(arr3, size);
    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Merge Sort]
└─$ g++ merge-sort-2.cpp

┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Merge Sort]
└─$ ./a.out
5
92
94
89
20
150
23
4325
2342
123
214
Time taken is 0.000247
Sorted array:
89 92 123 150 4325
Time taken is 0.000333
Sorted array:
20 23 94 214 2342
Time taken is 0.000337
Sorted array:
3333 168 93 86 54
```

Q3).

3. Modify the Merge-sort algorithm (discussed in the class) such that the algorithm takes the input as words (of different lengths) and arrange them in an alphabetical order. Your words will have both lower-case letters as well as the upper-case letters. Compute the time-complexity $t(n)$ of your algorithm in an experimental approach. Compare this algorithm with that of the algrotihm which takes n numbers as inputs and decide which consumes minimium time.

Code:

merge-string-sort.cpp > mergeSort(char [], int const, int const)

```cpp
#include <iostream>
#include <cstring>
#include <string>
#include <ctime>
#include<bits/stdc++.h>

using namespace std;

void merge(char array[], int const left, int const mid,int const right){
    auto const subArrayOne = mid - left + 1;
    auto const subArrayTwo = right - mid;
    auto *leftArray = new int[subArrayOne], *rightArray = new int[subArrayTwo];
    for (auto i = 0; i < subArrayOne; i++)
        leftArray[i] = array[left + i];
    for (auto j = 0; j < subArrayTwo; j++)
        rightArray[j] = array[mid + 1 + j];
    auto indexOfSubArrayOne = 0,
    indexOfSubArrayTwo = 0;
    int indexOfMergedArray = left;
    while (indexOfSubArrayOne < subArrayOne
        && indexOfSubArrayTwo < subArrayTwo) {
        if (leftArray[indexOfSubArrayOne] <= rightArray[indexOfSubArrayTwo]) {
            array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
            indexOfSubArrayOne++;
        }
        else {
            array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
            indexOfSubArrayTwo++;
        }
        indexOfMergedArray++;
    }
    while (indexOfSubArrayOne < subArrayOne) {
        array[indexOfMergedArray] = leftArray[indexOfSubArrayOne];
        indexOfSubArrayOne++;
        indexOfMergedArray++;
    }
    while (indexOfSubArrayTwo < subArrayTwo) {
        array[indexOfMergedArray] = rightArray[indexOfSubArrayTwo];
        indexOfSubArrayTwo++;
        indexOfMergedArray++;
    }
    delete[] leftArray;
    delete[] rightArray;
}
```

```cpp
void mergeSort(char array[], int const begin, int const end)
{
    if (begin >= end)
        return;
    auto mid = begin + (end - begin) / 2;
    mergeSort(array, begin, mid);
    mergeSort(array, mid + 1, end);
    merge(array, begin, mid, end);
}

void printArray(char A[], int size)
{
    for (auto i = 0; i < size; i++)
        cout << A[i] << " ";
}

int main()
{
    string s = "Sample String";
    const int length = s.length();
    transform(s.begin(), s.end(), s.begin(), ::tolower);
    char* char_array = new char[length + 1];
    strcpy(char_array, s.c_str());

    cout << "Given string is ";
    printArray(char_array, length);

    clock_t st=clock();

    mergeSort(char_array, 0, length - 1);

    cout << "\nSorted string is ";
    printArray(char_array, length);

    double time1=(double)(clock() - st)/CLOCKS_PER_SEC;
    cout<<"\n Time taken is "<<time1<<"s"<<endl;

    return 0;
}
```

Output:

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Merge Sort]
└─$ ./a.out
Given string is s a m p l e   s t r i n g
Sorted string is   a e g i l m n p r s s t
 Time taken is 1.3e-05s
```

The number of characters in the string is 13.

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Merge Sort]
└─$ ./a.out
Enter number of elements: 13
23
21
33
11
8
34
21
65
1
9
8
10
4
Given array is
23 21 33 11 8 34 21 65 1 9 8 10 4
Sorted array is
1 4 8 8 9 10 11 21 21 23 33 34 65
 Time taken is 1.2e-05s
```

The merge sort of numbers works better than the merge sort of alphabets.

Q8).

8. Modify the merge-sort algorithm in such a way that the left subarray (got as a result of the first divison) is divided further till we get a subarray of size 1 and the right-subarray (got as a result of the first division) is not subjected to any division. For sorting the right subarray, apply insertion-sort algorithm. Write the modified algorithm A' to sort the n-given numbers

Code:

merge-sort-8.cpp > main()

```cpp
#include<iostream>
using namespace std;
#include<vector>
#include <ctime>
#include<limits.h>
void merge(vector<int> &elements, int left, int mid, int right)
{
    vector<int> left_Sub_Array, right_Sub_Array;
    int no_Left_Sub,no_Right_Sub,i,index_Left_SA,index_Right_SA,index_Full_Array;
    no_Left_Sub = mid-left+1;
    no_Right_Sub = right-mid;
    for(i=0;i<no_Left_Sub;i++)
        left_Sub_Array.push_back(elements[left+i]);
    left_Sub_Array.push_back(INT_MAX);
    for(i=0;i<no_Right_Sub;i++)
        right_Sub_Array.push_back(elements[mid+i+1]);
    right_Sub_Array.push_back(INT_MAX);
    index_Left_SA=0;
    index_Right_SA=0;

    for(index_Full_Array=left;index_Full_Array<=right;index_Full_Array++)
    {
        if(left_Sub_Array[index_Left_SA] < right_Sub_Array[index_Right_SA]){
            elements[index_Full_Array] = left_Sub_Array[index_Left_SA];
            index_Left_SA++;
        }
        else{
            elements[index_Full_Array] = right_Sub_Array[index_Right_SA];
            index_Right_SA++;
        }
    }
}
```

```cpp
void mergesort(vector<int> &elements, int left, int right){
    int mid;
    if(left==right)
        return;
    mid = (left+right)/2;
    mergesort(elements,left,mid);
    mergesort(elements,mid+1,right);
    merge(elements,left,mid,right);
}

int main(){
    int n,i,ele;
    vector<int> elements;
    cin>>n;
    for(i=0;i<n;i++){
        cin>>ele;
        elements.push_back(ele);
    }
    clock_t tStart = clock();
    int mid=(n-1)/2;
    mergesort(elements, 0, mid);
    int m= n-mid+2;
    //cout<<m<<mid;
    for(int j=mid;j<m;j++){
        int key=elements[j];
        //cout<<elements[j];
        int i=j-1;
        while((i>=0)&&(key<elements[i])){
            elements[i+1]=elements[i];
            i=i-1;
        }
        elements[i+1]=key;
    }
    double time1=(double)(clock() - tStart)/CLOCKS_PER_SEC;
    cout<<"Time taken is "<<time1<<endl;
    for(i=0;i<n;i++){
        cout<<elements[i]<<"\t";
    }
}
```

Output:

```
┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Merge Sort]
└─$ g++ merge-sort-8.cpp

┌──(kali㉿kali)-[~/…/SEM4/DAA-Lab-Winter2023-main/IPS FILES/Merge Sort]
└─$ ./a.out
10
92
94
90
67
52
78
59
100
34
67
Time taken is 6e-06
52      59      67      78      90      92      94      100     34      67
```