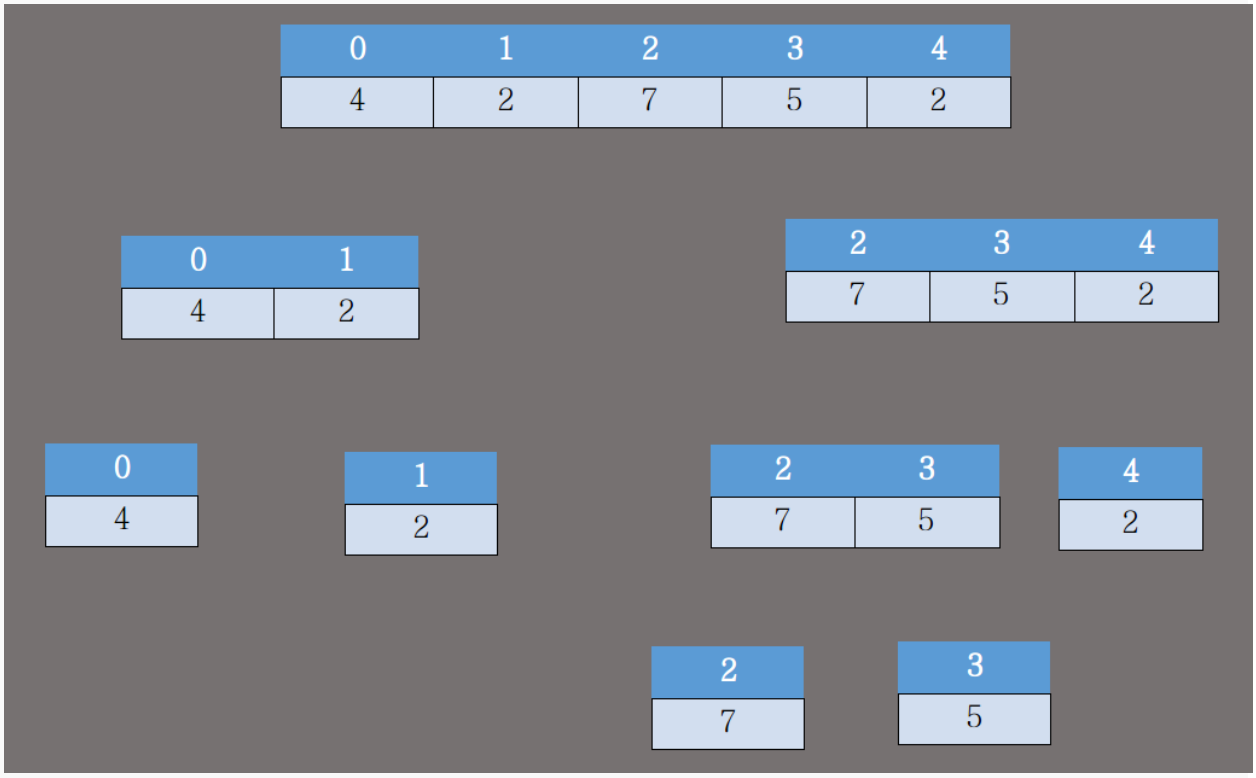
Small Large Split Merge Sort

Small Large Split Merge sort is a variation of standard merge sort, where the 'n' (if n>2) elements in the sequence are split into two parts with 2 elements in the first part and (n-2) elements in the second part. If n==2, then split them into two equal sorted sequences. Given the partial code of standard merge sort in the code editor, modify the code such that Small Large Split Merge Sort is implemented. There is no change in the merge function, here we merge function similar to the standard merge sort.

The idea of split is illustrated in the figure below:



1. Analyse the time complexity of this variant of merge sort and make a note in the algorithm part

2. Run the code (in your laptop) for different input sizes like 100, 200, ..., and plot a graph for the time taken by the algorithm

Input Format

First line contains the number of elements to be sorted, n

Next line contains the 'n' elements to be sorted which are separated by space

Output Format

Print the elements in ascending order

T(n)=O(n logn)

```
#include<iostream>
using namespace std;
#include<vector>
#include <ctime>
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;
     // Find number of elements in left subarray
     no_Left_Sub = mid-left+1;
     // Find number of elements in right subarray
     no_Right_Sub = right-mid;
     //make a copy of left sub array in a temporary array
     for(i=0;i<no_Left_Sub;i++)
          left_Sub_Array.push_back(elements[left+i]);
     //make a copy of right sub array into another temporary
array
     for(i=0;i<no_Right_Sub;i++)
          right_Sub_Array.push_back(elements[mid+i+1]);
     // Position the indices for the three arrays rightly
     index_Left_SA=0;
     index_Right_SA=0;
     index_Full_Array=left;
     // When there are elements to be comapred in both the
arrays
```

```
     while( (index_Left_SA<no_Left_Sub) &&
(index_Right_SA<no_Right_Sub))
     {
          // if element in left sub array is less than
element in right sub array
          if(left_Sub_Array[index_Left_SA] <
right_Sub_Array[index_Right_SA])
          {
               // copy the element from left sub array into
orginal array and increment indices
               elements[index_Full_Array] =
left_Sub_Array[index_Left_SA];
               index_Left_SA++;
               index_Full_Array++;
          }
          //otherwise
          else
          {
               // copy the element from right sub array into
orginal array and increment indices
               elements[index_Full_Array] =
right_Sub_Array[index_Right_SA];
               index_Right_SA++;
               index_Full_Array++;
          }
     }
     //If elements in left sub array are left over
     while(index_Left_SA<no_Left_Sub)
     {
               // copy the elements left over in left sub
array into orginal array and increment indices
               elements[index_Full_Array] =
left_Sub_Array[index_Left_SA];
               index_Left_SA++;
               index_Full_Array++;
     }
     //If elements in right sub array are left over
     while(index_Right_SA<no_Right_Sub)
```

```
    {
                // copy the elements left over in right sub
array into orginal array and increment indices
                elements[index_Full_Array] =
right_Sub_Array[index_Right_SA];
                index_Right_SA++;
                index_Full_Array++;
    }

  }

void mergesort(vector<int> &elements, int left, int right)

{

    if(left==right)

    {

        return;

    }

    if((right-left)==1)

    {

        if(elements[left]>elements[right])

        {

            int temp=elements[left];

            elements[left]=elements[right];

            elements[right]=temp;

        }

        return;

    }
```

```
    // call mergesort for left subarray
    mergesort(elements,left,left+1);
    // call mergesort for right subarray
    mergesort(elements,left+2,right);
    // merge the sorted left and right subarray
    merge(elements,left,left+1,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();
    mergesort(elements, 0, n-1);
    double time1=(double)(clock() -
tStart)/CLOCKS_PER_SEC;
```

```
    //cout<<"Time taken is "<<time1<<endl;
    for(i=0;i<n;i++)
    {
        cout<<elements[i]<<" ";
    }
}
```

Merge by Inserting

Merge by Inserting is a variation of merge sort in which we make change in the merge part of merge sort. The right part of the array is inserted into the left part of the array. That is the elements in the right part of the array has to be insered into the sorted array which is the left part of the array

For example, consider an array of elements with first four elements as left part and second four elements as right part. Left part and right part are sorted by themselves. In the merge step, insert the elements one by one in the right part into the left part. The left part will keep on increasing in length and will become whole array at the end. Consider the elements 1, 4, 7, 12, 2, 5, 8, 14.

Left part consist of 1, 4, 7, 12

Right part consist of 2, 5, 8, 14

Insert 2 into sorted left part, then 5, 8 and 14

1. Analyse the time complexity of this variant of merge sort and make a note in the algorithm part

2. Run the code (in your laptop) for different input sizes like 100, 200, ..., and plot a graph for the time taken by the algorithm

Input Format

First line contains the number of elements to be sorted, n

Next line contain the 'n' elements to be sorted (separated by a space)

Output Format

Print the 'n' elements in ascending order

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

```
#include<iostream>

using namespace std;

#include<vector>

#include <ctime>

void merge(vector<int>& elements, int left, int mid, int
right)

{

    //insert the elements in the right part into the left
part

    int n=right-left+1;

    for(int i=n/2;i<n;i++)

    {

        int key=elements[i];

        int j=i-1;

        while(j>=0 && elements[j]>key)

        {

            elements[j+1]=elements[j];

            j--;

        }

        elements[j+1]=key;
```

```
    }
}
void mergesort(vector<int> &elements, int left, int right)
{
     int mid;
     // When there is only one element in the array
     if(left==right)
          return;
     // Find mid of the array
     mid = (left+right)/2;
     // call mergesort for left subarray
     mergesort(elements,left,mid);
     // call mergesort for right subarray
     mergesort(elements,mid+1,right);
     // merge the sorted left and right subarray
     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();
    mergesort(elements, 0, n-1);
    double time1=(double)(clock() -
tStart)/CLOCKS_PER_SEC;
    //cout<<"Time taken is "<<time1<<endl;
    for(i=0;i<n;i++)
    {
        cout<<elements[i]<<" ";
    }
  }
```