Week 3

Students are advised to refer the class discussion for further understanding

Unit 3: Part II: Divide and Conquer

- **Principle:**
- Input is divided into small groups before processing using logical division.
- Solution space for all the divided groups is same and unique.
- The CPU utilization is on higher side whereas the throughput is on lower side.
- The division of input reduces the complexity of the process and most of DAC based algorithms have logarithmic complexity.
- DAC principle is mostly applied for input of large size.

Binary Search

- **Principle:**
- Implemented on the SORTED array
- Input is divided into small groups before processing using formula:

$$mid = \frac{low+high}{2}$$ (Consider lower integer value)

- Where: "low" represents start of array and "high" represents end of array.
- If element is found at position "mid" search is successful, otherwise compare the element at "mid" and element to be searched.
- If a[mid] < elem ➔ low = mid + 1
- If a[mid] > elem ➔ high = mid -1
- Continue process till element is found (a[mid] = elem)
- Or otherwise if (low > high) ➔ Search is unsuccessful.

Binary Search: Numerical on average number of comparisons for successful and unsuccessful search

- Consider the following array of size: 14 (1..14 ➔ low=1 and high=14)

I	1	2	3	4	5	6	7	8	9	10	11	12	13	14
A	-10	-5	7	14	21	34	45	56	65	77	89	98	104	110
C	3	4	2	4	3	4	1	4	3	4	2	4	3	4

- Total number of comparisons for successful search: 43
- Average number of comparisons = 43/14 = 3.07
- For finding total number of comparisons for unsuccessful search:
  - Draw Binary search tree for the given array
  - Complete the BST and find the sum of level of BST
  - Compute average by dividing the sum/n+1 (where n = number of elements)

Binary Search: Algorithm and complexity equation

- ***Algorithm BinarySearch(a,i,l,x) //x is the element to be searched***

```
{
        If(i==l) then
        {
                If(x=a[i])
                 return(i);
                else
                 return(0);
        }

        Else
        {
                Mid=[i+l]/2
                If(x==a[mid]) then
                   Return(mid)
                Else
                If(x<a[mid])then
                        Return (bsearch(a,i,mid-1,x)
                Else
                        Return (bsearch(a,mid+1,l,x);
        }
```

Ternary Search

```
Ternary Search:
Algorithm ternarySearch(a, low, high, searchValue)
{
                int left = 1;
                int right = n;
                while(left < right)
                {
                        int middle1 = (right + left) / 3;
                        int middle2 = (right + left) / 3 * 2;

                if(a[middle1] == searchValue)
                {
                    return middle1;
                }
                else if(a[middle2] == searchValue)
                {
                return middle2;
                 }
                else if (searchValue < a[middle1])
                {
                right = middle1 - 1;
                }
                else if (searchValue > a[middle1] && searchValue < a[middle2])
                {
                                left = middle1+1;
                                right = middle2 - 1;
                 }
                else if (searchValue > a[middle2])
                                left = middle2;
                }
                }

}
```

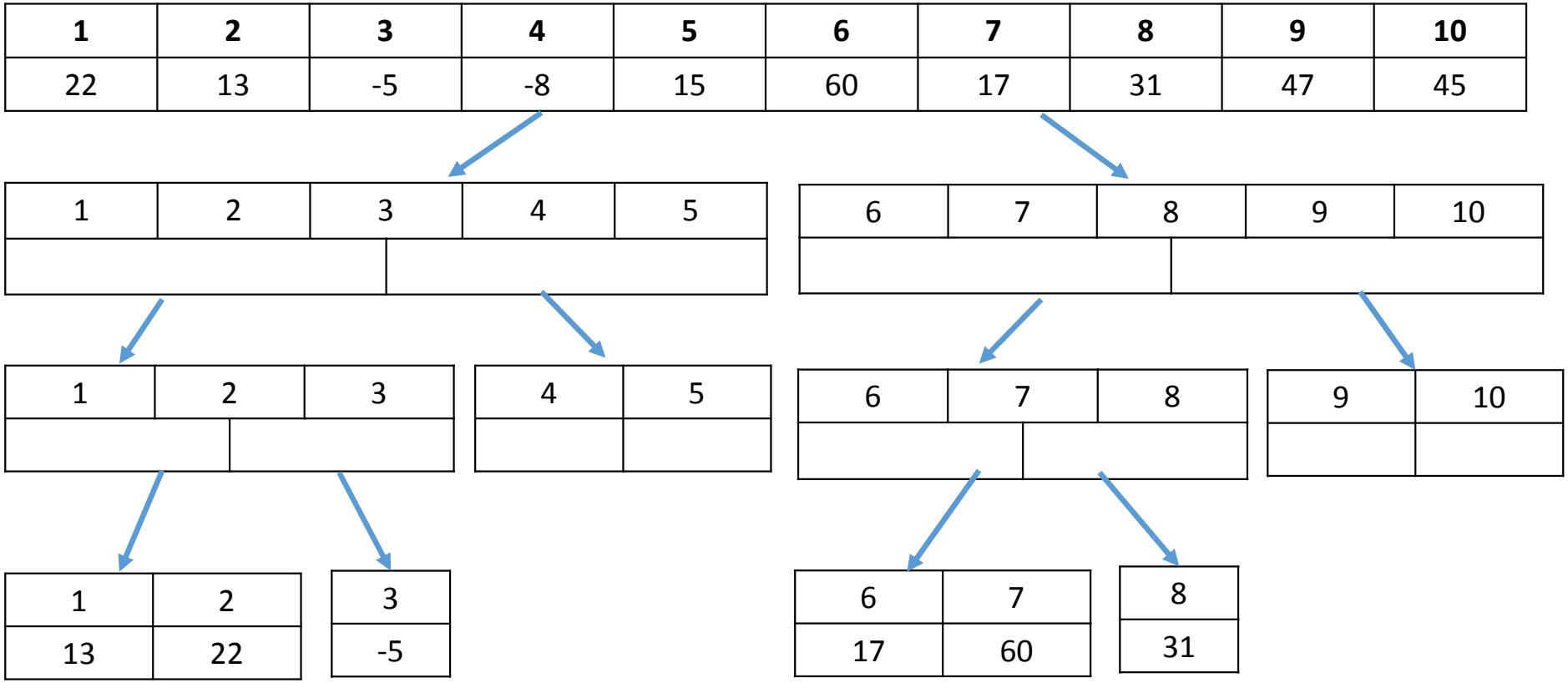
Min-Max Algorithm

- **Principle:** Method to find minimum and maximum element from an array.
- General method requires 2*(n-1) comparisons.

Algorithm min_max(a,n,min,max)

```
{
    min = max = a[1]
    for i = 2 to n do
    {
       if (a[i] < min) then min = a[i]
       if (a[i] > max) then max = a[i]
    }
}
```

Min-Max: DAC based Example



Min-Max: DAC Algorithm

```
Algorithm min_max(a,i,j,min,max)
{
If (i==j) then min=max=a[i]
Else
If(i=j-1) then
{
   if(a[i] < a[j]) then
   {
      min = a[i]; max=a[j]
    }
    else
   {
      min=a[j]; max=a[i]
   }
Else
  {
      mid = [i+j] / 2 (Take lower integer)
      min_max(a,i,mid,min,max)
      min_max(a,mid+1, j, min1, max1)
      if (max < max1) then max = max1
      if (min > min1) then min = min 1
   }
}
```

- Complexity equation:
- T(n) = T(n/2) + T(n/2) + 2 for n > 2
- T(n) = 1 if n=2
- T(n) = 0 if n=1