Sorting

Objective

The objective of this lab is to implement sorting algorithms using array-based implementation.

Task

1. Quicksort is a divide and conquer algorithm. Quicksort first divides a large array into two smaller sub-arrays using pivot value. Quicksort then recursively sorts the sub-arrays by repeating the same process of partition on each sub-list. The average case of quicksort is O(n log n) when the pivot is nearly median, while the worst case is O(n^2) when the pivot is the smallest or largest item of the list.
   Implement the quick sort algorithm using recursion. Call partition method for each sub-list.

Procedure for Quick Sort

- Create an array of random numbers and pass it to the quicksort method.
- Consider the first element as a pivot for each sub-list of a quick sort algorithm.
- Count the number of comparisons and swapping to understand the complexity.

```
public static void main(String[]args) {

     int[] arr=new int[100];

    for(int i=0;i<100;i++){
        arr[i]=(int)(Math.random()*100)+1;
    }

      QuickSort(arr, 0,arr.length-1);

    // print arr
    for(int i=0;i<100;i++){
        System.out.print(arr[i]+", ");
      }
}
```

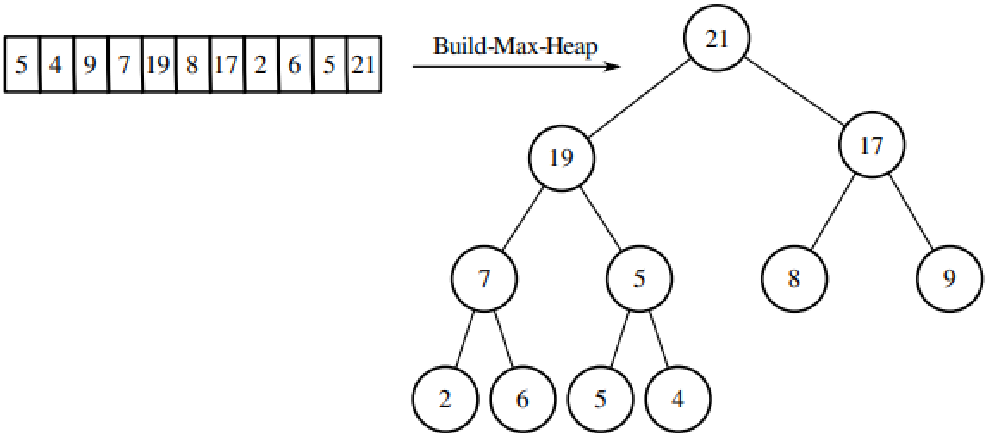
```
Class SortingAlgorithm{

public void QuickSort(int[] arr, int L, int U){
  int m=Partition(arr,L,U);
      if(m-1>L){
            QuickSort(arr,L,m-1);
      }
      if(m+1<U){
            QuickSort(arr,m+1,U);
  }

public int Partition(int[] arr, int L, int U){
    //This function consider the first element as
           pivot, places all smaller (smaller than
           pivot) to the left of pivot and all greater
           elements to right of pivot.
              }
}
```

Task

2. Write a program that fills an array of n elements by randomly generated Integers, range (1-100), and sorts them in ascending order using heap sort.
    a. Create a Max-heap tree using array-based implementation by swapping the data in an array.



    b. To perform heap sort, delete a max value from a heap tree and replace it with the last element of the array, then re-heap the tree by reducing the list size by one to make it again a max-heap.

Procedure Heap Sort

a. In an array-based implementation, compute left child and right child using the formula 2P+1 and 2P+2. While to compute parent of a node c use formula (c-1)/2.
Note: this method will call only once to create a first heap tree

Algorithm to buildMAxHeap:

```
1. First child c starts from index=1 to index<arr.length
2. For each child c calculate its parent p
3. While(p>=0)
       ▪ If (c value is greater than parent p-value )
             • then swap c and p values in arr.
       ▪ Update c and p
       ▪ Continue to step 3
4. Continue to step 1 to repeat the same for each child.
```

b. To perform heapsort, delete the max value from the tree and re-heap the tree again on the remaining elements. Do not use the build-heap method here.

Algorithm to delete:

1. Find the node to delete a node from the heap tree.
2. Replace the deletion node with the "farthest node" on the lowest level of the Binary Tree. (or last element of an array).
3. Delete the farthest node (last element).
4. Re-Heap (fix the heap):
   - Compute child of replacement node, left-child, and right-child
   - Filter the replacement node DOWN the binary tree by moving up the greater child.

```
Class HeapTree{


        public void buildMaxHeap(int[] arr){  ... }

        public void ReHeap(int[] arr){  ... }


        public void delete(){
        //always delete root

        … }



}
```