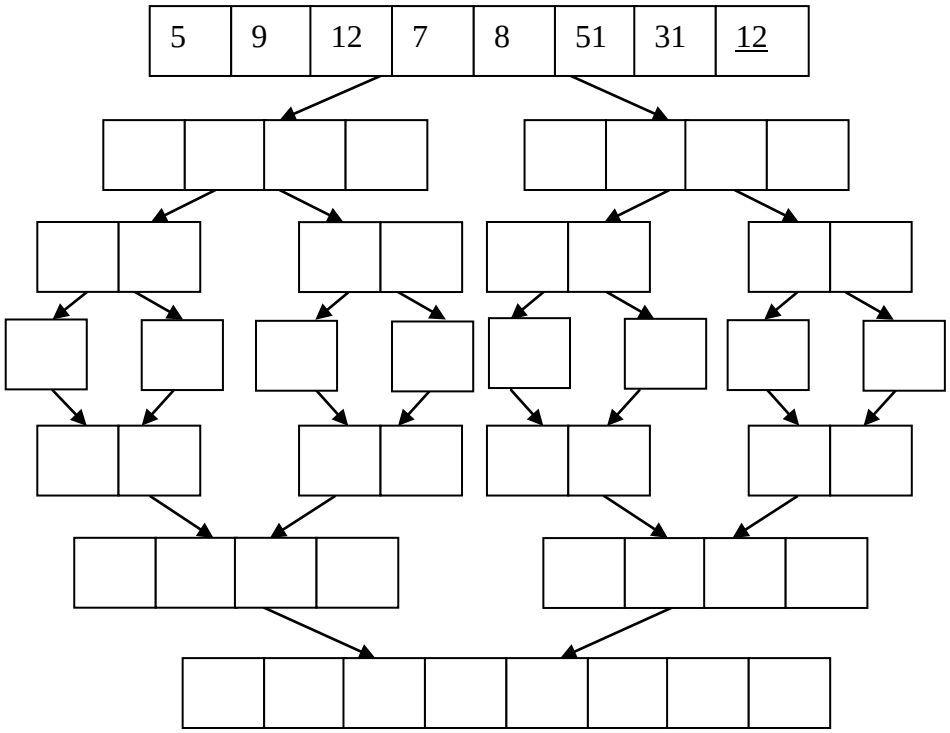
CS1102: Data Structure and Algorithms

Tutorial 7: Sorting

Week of 15 March 2010

1. a. Perform merge sort on the following array:



b. Merge sort has the following recursive structure:

```
1. public static void mergeSort(int[] a, int i, int j){
2. if (i < j) {
3.         int mid = (i+j)/2;
4.         mergeSort(a,i,mid);
5.         mergeSort(a,mid+1,j);
6.         merge(a,i,mid,j);
7.         }
8. }
```

Suppose line 6 is removed, what is the time complexity of the modified program?

CS1102: Data Structure and Algorithms

2. Based on the Big-O notation, which of these sorting algorithms would you use to sort the following data stored in an array: (1) Merge Sort; (2) Quick Sort with first element pivot; (3) Insertion Sort; (4) Selection Sort; (5) Bubble Sort (improved version) and (6) Radix Sort? Explain your answer.

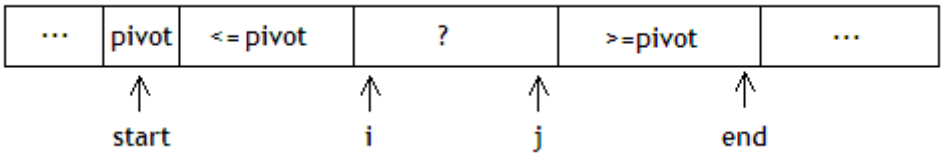
a. 1,000,000 distinct integers from 0 to 999,999 in reverse order.
b. 1,000,000 distinct real numbers from 0.0 to 1.0 in random order.
c. 1,000,000 distinct integers from 0 to 999,999 with only one element out of place, i.e. the array is sorted without this element.
d. 1,000,000 distinct real numbers from 0.0 to 1.0, where all the elements are at most 5 places away from their proper position.

3. Write Java methods to achieve the following tasks. Your methods should have the complexity of O (n log n):

a. Find the median of an integer array (Recall that the median of a list of n number is defined as its $\lceil n/2 \rceil$ smallest element). Assume our array is not empty.
b. Check whether an integer array contains duplicates.
c. Find the distance between the two closest numbers in an integer array.

4. This question will refer to the *QuickSort* algorithm stated in your lecture notes.

a. Implement the *QuickSort* algorithm by using two indices ***i*** and ***j*** as shown in the diagram below to iterate through the subarray during each recursive call to find the partition index.




```
void QuickSort(int[]A, int start, int end)
{
  ...
}
```

b. Since *QuickSort* has a time complexity of O(n^2) in the worst case, and O(*n log n*) in the best/average case, while merge sort has a time complexity of O(*n log n*) under all cases, explain why we still use *QuickSort* instead of *MergeSort*.