Cost of Max Array Problem

There're couple ways to find the maximum value of the array:

1. Sort array and pick the first element: O(nlogn) + O(1) = O(nlogn)
2. Pairwise comparisons: O(n), since it's scanning the entire array and stores the maximum element from the element scanned so fat at each step.
3. Divide & compare: This HW is to discuss the cost of this recursive division and conquer method.

Let's say there's an array:

arr1 = [5,3,7,4,9,6] n =6
arr2 = [5,3,7,4,9,6,2,8,1] n= 9

1.Split in half until there are only arrays with 1 or 2 elements:

For arr1

After 1st split: [5 3 7], [4, 9, 6]
After 2nd split: [5],[3,7],[4],[9,6]
// 1 + 2 = 3 splits

For arr2

After 1st split: [5,3,7,4,9],[6,2,8,1]
After 2nd split: [5,3,7],[4,9],[6,2],[8,1]
After 3rd split: [5],[3,7],[4,9],[6,2],[8,1]
// 1 + 2 +1 = 4 splits

If splitting an array in half cost O(1), the total cost for all division would be n/2 * O(1) = O(n)

2. Compare and drop the lower element, then merge:

For arr1

[5],[3,7],[4],[9,6]

After 1st drop: [5],[7],[4],[9]
After 1st merge: [5,7],[4,9]
// 2 comparison, 2 drop, 2 merge

After 2nd drop: [7],[9]
After 2nd merge: [7,9]
// 2 comparison, 2 drop, 1 merge

After 3rd drop: [9]
// 1 comparison, 1 drop

For arr2

[5],[3,7],[4,9],[6,2],[8,1]

After 1st drop: [5],[7],[9],[6],[8]
After 1st merge: [5,7],[9,6],[8]
// 4 comparison, 4 drop, 2 merge

After 2nd drop: [7],[9],[8]
After 2nd merge: [7,9],[8]
// 2 comparison, 2 drop, 1 merge

After 3rd drop: [9],[8]
After 3rd merge:[9,8]
// 1 comparison, 1 drop, 1 merge

After 4th drop: [9]
// 1 comparison 1 drop

Assume 1 comparison, drop and merge cost O(1) each:
Since the number of merge operation should be the same as split operation, so merge costs O(n)

For comparison and drop operations, at the base level (1st comparison and drop), they are operated at most n/2 times, and at the second level, at most n/4 times, and so on until n/(n^x) = 1
So the total cost would be (n/2 + n/4 + … + 1) * 2* O(1) = (n/2+ n/4 + … 1) * O(n)

Assume $n = 2^{k+1}$,

$$\frac{n}{2} + \frac{n}{4} + \ldots + 1 = 2^k + 2^{k-1} + \ldots + 1 = 2^{k+1} - 1 = n - 1$$

Hence, the total cost of the comparison and drop would be (n-1) * O(1) = O(n)

Combining the split, comparison&drop, merge operation costs, the total is: O(n) + O(n) + O(n) = O(n)