Q1 (1 point): Consider the following sorting algorithm. If the input length is even, run insertion sort. Otherwise, run Mergesort. Suppose $T(n)$ denotes the worst-case running time of this algorithm. Which is true?

- [✓ (A) $T(n) = O(n^2)$ but not $\Omega(n^2)$.]
- (B) $T(n) = \Theta(n^2)$.
- (C) $T(n) = O(n \log_2 n)$.

Answer:
$T(n)$ = worst Case

Since it runs insertion Sort on even elements it will never be less than n^2. By the definition,
$\exists\ K>0, \exists\ n_0\ \forall n > n_0\ f(n) \in O(n^2)$
The upper bound of $f(n)$ will be n^2.

Q2 (1 point): Answer True or False. $\sum_{i=0}^{n} 2^i = \Theta(\sum_{i=0}^{n} 3^i)$
Answer:

$$f(n) = \sum_{i=0}^{n} 2^i = \frac{2^{n+1}-1}{2-1} = 2^{n+1}-1 \quad g(n) = \frac{3^{n+1}}{3-1} = \frac{3^{n+1}-1}{2}$$

$$\lim_{n\to\infty} \frac{f(n)}{g(n)} = \frac{(2^{n+1}-1)\cdot 2}{3^{n+1}-1} = \lim_{n\to\infty} 2 \cdot \frac{2^{n+1}-1}{3^{n+1}-1}$$

$$= 2 \lim_{n\to\infty} \frac{\frac{2^{n+1}}{3^{n+1}} - \frac{1}{3^{n+1}}}{\frac{3^{n+1}}{3^{n+1}} - \frac{1}{3^{n+1}}}$$

$$= 2 \cdot \lim_{n\to\infty} \frac{0-0}{1-0}$$

$$= 2 \cdot 0 = \emptyset$$

Therefor

$$\sum_{i=0}^{n} 2^i \neq \Theta\left(\sum_{i=0}^{n} 3^i\right)$$ False.

Q3 (1 point): Consider an array A of positive integers. You need to reorder the array such that all the odd numbers appear first, and then the even numbers. You can only use $O(1)$ additional storage, so no hash tables allowed.

Give a short description of your algorithm, and the worst case running time. You should not need more than five lines to explain your answer.

Answer:

Brute force: 2 for loops traversing the array finding all odd numbers and swapping them with even numbers if the position of even is before odd.

This will be $O(n^2)$ since there is 2 for loops traversing the array.

Pivoting: Pointer in front (i) and pointer in rear (j). if ~~f~~ i is odd then i++
if j is even then j--
if i is even and j is odd,
Swap i and j

This should take $O(n)$ because in the worst case scenerio, say no even numbers than it will traverse the entire array making it $O(n)$.

Q4 (2.33 points): We discussed the algorithm to merge two sorted arrays. Give an algorithm to merge k sorted arrays. For convenience, assume that every array has size exactly n. Your running time will depend on both k and n.

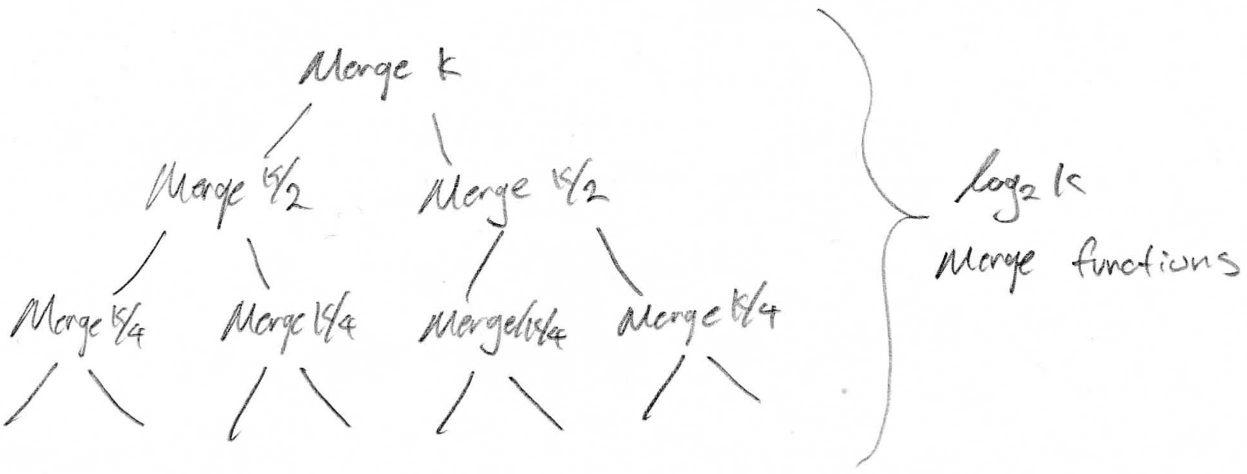
Answer:

Brute force: Com pare every Array (Merge(1,2,3,..., k))

This will take $O(n \cdot k)$

Divide and Conquer:

Split the k arrays into halves and contine to reccursively split the k arrays into halves and merge accordingly.



Since arrays are n lengths long and $\log_2 k$ Merge functions it will be $\underline{O(n \log_2 k)}$

Q5 (2.33 points): A sorting algorithm is called *stable* if the relative ordering of *equal* elements does not change. Thus, for $i < j$, if $A[i] = A[j]$, after a stable sort, the element initially at $A[i]$ ends up before the element initially at $A[j]$.

For example, given $A = [4\ 3\ 10\ 8\ 4]$, the sorted version is $B = [3\ 4\ 4\ 8\ 10]$. In a stable sort, the element $A[0]$ (which is 4) will end up as $B[1]$, and the element $A[4]$ (which is also 4) will end up at $B[2]$. In a stable sort, the two "4"s will not switch their order.

(This is relevant when sorting an array A of objects, by an integer field (say $A[i].key$). In a stable sort, the following is always true: if $i < j$ and $A[i].key = A[j].key$, then after sorting, the object initially at $A[i]$ will precede the object initially at $A[j]$.)

Show that Mergesort (with a very simple modification/clarification) is stable. Show that Quicksort is not stable, by providing an example.

Answer:

Merge Sort:

See next page.

uickSort:

quick Sort is not stable because even with a sorted array

ne elements can moe if you choose a random element, but

is even more prevelant with an unsorted array.

g, | 42 | 48 | 3 | 8 | 2 | 3 |

| 3 | 3 | 8 | 2 | 42 | 48 | Since the 3's swapped

| 2 | 3 | 3 | 8 | 42 | 48 | Position it is not stable.

Merge Sort.

nvariant: The "divide" of merge Sort will not Change the indeces of the elements, and Merge is Stable.

Base Case (n=2)

$$\begin{tikzcd} & (4,4) \arrow[dl] \arrow[dr] & \\ 4 \arrow[dr, dash] & & 4 \arrow[dl, dash] \\ & \boxed{4\ 4} & \end{tikzcd}$$

Proof:

When merge Sort seperates into halves, nothing actually happens to them when they Seperate. The indeces do not Change. When the merge Operation takes place, we can ensure that merge is Stable as long as it takes the "left" array when elements are equal.

Q6 (2.33 points): Given an array A, give pseudocode and big-Oh running time analysis for an algorithm that determines which permutation of the *stable* sorted order is A. Meaning, output an array P, such that $P[i]$ is the index of the element $A[i]$ after a stable sort of A.

For example, given $A = [4\ 3\ 10\ 8\ 4]$, the output should be $P = [1\ 0\ 4\ 3\ 2]$. So $A[0] = 4$ and it is at index 1 in the sorted order. Thus, $P[0] = 1$, etc. Note that because of the stable sorting, the last 4 goes to index 2 (and thus $P[4] = 2$). Furthermore, $A[1] = 3$, which ends up at index 0 in the sorted order. Thus, $P[1] = 0$. So on and so forth. (Yeah, my permutations start from 0, instead of 1, just to keep indexing easier.)

You may assume access to a stable sorting algorithm, like the version of Mergesort you designed in Q5. If the whole discussion about stability is confusing you, solve this problem assuming all elements are distinct. (In this case, the stability is irrelevant.) You will get partial credit.

Answer:

Selection Sort:

Since it finds the smallest element first as long as it finds the smallest element and not switch the first smallest element found, it should be stable and is $O(n^2)$

Merge Sort:

We make an array B with the indexes 0 to n-1 So.. $B = [0, 1, 2, \ldots, n-1]$ and we use merge sort to sort A using B... So $A[B[0]]$ to find its value. So we sort B while indexing array A when is unsorted. therefor by the end we will have a "sorted" A array with the indexes saved in B. We assumed that merge sort is stable by Q5. and hence proven.

This takes $O(n \log n)$ time.