1 What is the running time of the *QuickSort* algorithm, with the naive implementation of *ChoosePivot*, when the n-element input array is already sorted?

1. $\Theta(n)$
2. $\Theta(n \cdot logn)$
3. $\Theta(n^2)$
4. $\Theta(n^3)$

(3) is the correct answer. When we always choose the left most element of the array we get a recurrence of the form $T(n) = T(n-1) + O(n)$. This recurrence will evaluate to $T(n) = O(n^2)$.

2 What is the running time of the *QuickSort* algorithm, with the overkill implementation of *ChoosePivot*, on an arbitray input array? Assume that the *ChoosePivot* subroutine runs in $\Theta(n)$ time.

1. Insufficient information to answer
2. $\Theta(n)$
3. $\Theta(n \cdot logn)$
4. $\Theta(n^2)$

(4) is the correct answer. If we obtain a perfect split we get a recurrence of the form $T(n) = 2 \cdot T(\frac{n}{2}) + O(n)$, this recurrence evaluates to $\Theta(n \cdot logn)$.

3 Fix two different elements of the input array, say z_i and z_j. How many times might z_i and z_j be compared with each other during the execution of *QuickSort*?

1. exactly once
2. 0 or 1 times
3. 0, 1, or 2 times
4. any number between 0 and $n-1$ is possible

(2) is correct. We show why the other solutions are wrong to provide more insight. (1) is wrong because there is possibility that in fact the two elements are never compared. For example if the one of the fixed elements is the largest element in the array and the other fixed element is the smallest element in the array, if we chose any other element as the pivot those elements will never be compared to one another. (3) and (4) is wrong because we can show that in fact

the largest number of times any two elements can be compared to one another in the execution of *QuickSort* is once. To see this we note that two elements are compared to one another only if one of those elements are the pivot. Once they have been compared the pivot element is never compared to another element. From this it is the case two elements can be compared to each other at most once.

4 Recall the *Partition* subroutine employed by *QuickSort*. You are told that the following array has just been partitioned around some pivot element $\begin{bmatrix}3 & 1 & 2 & 4 & 5 & 8 & 7 & 6 & 9\end{bmatrix}$. Which of the elements could have been the pivot element?

4 and 5 are both the possible pivots. They are the only two entries in the such all elements to the left of them are smaller than them and all elements to right of them are larger than them.

5 Prove the correctness of the *QuickSort* algorithm

Let $P(n)$ denote the statement "for every input array of length n, *QuickSort* correctly sorts it." The base case ($n = 1$) is trivially true. We assume $P(i)$ holds for all $i = 1, 2, 3, \ldots, k-1$, we now show that $P(k)$ must be true. The partitioning subroutine places the pivot element p in the correct position it would be in the sorted array and both recursive calls are on arrays of length at most $n-1$, by $IHOP$ both of these will be sorted correctly. This concludes the correctness of *QuickSort*

5 Show that for every input array of length $n \geq 1$, the expected number of comparisons between input array elements in randomized *QuickSort* is at most $2 \cdot (n-1) \cdot lnn = O(n \cdot logn)$.

Let $C(\omega)$ be the number of comparisons given the pivot sequence ω. Let X_{ij} be that the ith and jth indices. Then $C(\omega) = \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} X_{ij}(\omega)$.

$$\begin{aligned}
E[C] &= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} E[X_{ij}] \\
&= \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} Pr[X_{ij} = 1] \\
&= \sum_{i=1}^{n-1} \frac{2}{j-i+1} \\
&\leq \sum_{i=1}^{n-1} \sum_{j=2}^{n} \frac{2}{j} \\
&= 2 \cdot (n-1) \cdot \sum_{j=2}^{n} \frac{1}{j} \\
&\leq 2 \cdot (n-1) \cdot \int_{j=1}^{n} \frac{1}{j} dj \\
&= 2 \cdot (n-1) \cdot lnn
\end{aligned}$$

6 Let α be some constant, independent of the input array length n, strictly bettween 0 and $\frac{1}{2}$. What is the probability that with a randomly chosen pivot element, the *Partition* subroutine produces a split in which the size of both the resulting subproblems is at least α times the size of the input array?

1. α
2. $1 - \alpha$
3. $1 - 2 \cdot \alpha$
4. $2 - 2 \cdot \alpha$

(3) is the correct answer. Since both problems must be at least $\alpha \cdot n$ then the pivot must be greater than or equal to $\alpha \cdot n$. Similarily the pivot must also be less than or equal to $n - \alpha \cdot n$ since if not the rhs would have subarray smaller than $\alpha \cdot n$. From this we deduce pivot,denote p must be between both $\alpha \cdot n$ and $n - \alpha \cdot n$. Hence we get that $Pr(\alpha \cdot n \leq p \leq n - \alpha \cdot n)$ is the probability

we are looking for. Since we have a uniform probability distribution the sum is expressed as

$$\begin{aligned}\sum_{\alpha\cdot n}^{n-\alpha\cdot n}\frac{1}{n} &= \frac{1}{n}\cdot\sum_{\alpha\cdot n}^{n-\alpha\cdot n}1\\ &= \frac{1}{n}\cdot[n-\alpha\cdot n-\alpha\cdot n]\\ &= \frac{n-2\cdot\alpha\cdot n}{n}\\ &= 1-2\cdot\alpha\end{aligned}$$

7 Let α be some constant, independent of the input array length n, strictly between 0 and $\frac{1}{2}$. Assume you achieve the approximately balanced splits from the preceding problem in every recursive call-so whenever a recursive call is given array of length k, each of its two recursive calls is passed a subarray with length between $\alpha\cdot k$ and $(1-\alpha)\cdot k$. How many successive recursive calls can occur before triggering the base case? Equivalently, which levels of the algorithm's recursion tree can contains leaves? Express your answer as a range of possible numbers d, from the minimum to the maximum number of recursive calls that might be needed.

(2) is correct.

1. $0 \le d \le -\frac{lnn}{ln\alpha}$
2. $-\frac{lnn}{ln\alpha} \le d \le -\frac{ln}{ln(1-\alpha)}$
3. $-\frac{lnn}{ln(1-\alpha)} \le d \le -\frac{lnn}{ln\alpha}$
4. $-\frac{lnn}{ln(1-2\cdot\alpha)} \le d \le -\frac{lnn}{ln(1-\alpha)}$

8 Define the recursion depth of *QuickSort* as the maximum number of successive recursive calls it makes before hitting the base case-equivalently, the largest level of its recursion tree in randomized *QuickSort*, the recursion depth is a random variable, depending on the pivots chosen, What is the minimum and maximum-possible recursion depth of randomized *QuickSort?*

1. minimum: $\Theta(1)$; maximum: $\Theta(n)$
2. minimum: $\Theta(logn)$; maximum: $\Theta(n)$
3. minimum: $\Theta(logn)$; maximum: $\Theta(n\cdot logn)$

4. minimum: $\Theta(\sqrt{n})$; maximum: $\Theta(n)$

(2) is the correct answer. As described in text the expected running time of *QuickSort* is $O(n \cdot logn)$.

9 Extend the $\Omega(n \cdot logn)$ lower bound in to apply also to the expected running time of randomized comparison based sorting algorithms.

We restate the proof that a deterministic algorithm is $\Omega(n \cdot logn)$ for completeness sake. Assume that we are sorting a list of n distinct numbers. Since the leaves corresponds to all possible ordering there must be $n!$ leaves. We also know that a tree with height h must have at most 2^h leaves. From this we obtain the inequality

$$\begin{aligned}
2^h &\geq n! \\
h &\geq log_2(n!) \\
&\geq log_2(\Pi_{i=1}^{n} i) \\
&\geq log_2(1) + log_2 2 + \ldots + log_2(n) \\
&\geq \sum_{i=2}^{n} log_2(i) \\
&= \sum_{i=2}^{\frac{n}{2}-1} log_2(i) + \sum_{i=\frac{n}{2}}^{n} log_2 i \\
&\geq \sum_{i=\frac{n}{2}}^{n} log_2 i \\
&\geq \sum_{i=\frac{n}{2}}^{n} log_2 \frac{n}{2} \\
&= \frac{n}{2} \cdot log_2 \frac{n}{2} \\
&= \Omega(n \cdot log_2 n)
\end{aligned}$$

From this we can use Yao's minimax principle, since $\Omega(n \cdot log_2 n)$ is a lowerbound for deterministic sorting algorithms, $\Omega(n \cdot log_2 n)$ is also a lowerbound on any Las Vegas randomized algorithm.