CLRS 5.2. Searching an unsorted array (pg. 144)

Input: A sequence of n numbers $A = \langle a_1, a_2, \ldots, a_n \rangle$ and a value v
Output: An index i s.t. $v = A[i]$ or the special value NIL if v does not appear in A.

```
DETERMINISTIC-SEARCH(A, v):
1  for i := 1 to A.length
2      if A[i] = v then
3          return i
4  return NIL
```

```
SCRAMBLE-SEARCH(A, v):
1  A' := PERMUTE(A)
2  return DETERMINISTIC-SEARCH(A', v)
```

g. Suppose that there are no indices i such that $A[i] = x$. What is the average-case running time of DETERMINISTIC-SEARCH? What is the worst-case running time of DETERMINISTIC-SEARCH?

Given the fact that the array does not contain the element we are looking for, the average-case execution coincide with the worst-case execution, i.e. in both cases, the algorithm will check all indices before terminating. Assuming that permuting an array is an $\Theta(n)$ operation, we get

$$T_{det}^{avg}(n) = T_{det}^{worst}(n) = \Theta(n)$$

h. Letting k be the number of indices i such that $A[i] = x$, give the worst-case and expected running times of SCRAMBLE-SEARCH for the cases in which $k = 0$ and $k = 1$. Generalize your solution to handle the case in which $k \geq 1$.

If $k = 0$, we have the same situation as in g.: The average-case execution coincide with the worst-case execution, the running time is $\Theta(n)$ in both cases.

If $k = 1$, let us introduce an indicator random variable representing the number of comparisons we need to make:

$$X_i = \begin{cases} 1 & \text{if } A'[i] = x \quad \text{i.e. } i \text{ comparisons are needed} \\ 0 & \text{otherwise} \end{cases}$$

When calculating the expected value, we can use the fact that we only need to fix one element (i.e. the i'th element is x), the rest can be in any permutation.

$$E[X_i] = P(X_i = 1) = \frac{(n-1)!}{n!} = \frac{1}{n}$$

If X represents the number of comparisons we need to make, then

$$X = \sum_{i=1}^{n} i \cdot X_i$$

From this we can directly calculate the expected value

$$E[X] = E\left[\sum_{i=1}^{n} i \cdot X_i\right] = \sum_{i=1}^{n} i \cdot E[X_i] = \sum_{i=1}^{n} i \cdot \frac{1}{n} = \frac{(n+1)}{2}$$

The worst case is when the element we are looking for happens to be the last one. In this case, the running time is n.

For $k \geq 1$, let us introduce another indicator random variable representing the number of comparisons we need to make:

$$Y_i = \begin{cases} 1 & \text{if } A'[i] = x \text{ and } \forall j < i: A'[j] \neq x \\ 0 & \text{otherwise} \end{cases}$$

To have $Y_i = 1$, we first have to

1. Pick $i - 1$ elements that are not x: $P(n-k, i-1)$
2. Pick one of the x's: $C(k, 1)$
3. Pick a permutation of the remaining elements: $P(n-i, n-i)$
4. Divide by $k!$ to avoid counting identical permutations multiple times.

$$E[Y_i] = P(Y_i = 1) = \frac{P(n-k, i-1) \cdot C(k, 1) \cdot P(n-i, n-i) / k!}{n! / k!}$$

$$= \frac{\frac{(n-k)!}{(n-k-i+1)!} \cdot k \cdot (n-i)!}{n!}$$

$$= \frac{k(n-k)!}{n!} \cdot \frac{(n-i)!}{(n-k-i+1)!}$$

(We assumed that other elements are all unique. However, this assumption is not necessary, as the corresponding divisors appear in both the numerator and the denominator.)

Let Y be a random variable corresponding to the number of indices we must pick.

$$Y = \sum_{i=1}^{n} i \cdot Y_i$$

Using the linearity of expected value, we get

$$\begin{aligned} E[Y] = E\left[\sum_{i=1}^{n} i \cdot Y_i\right] &= \sum_{i=1}^{n} i \cdot E[Y_i] \\ &= \sum_{i=1}^{n} i \left(\frac{k(n-k)!}{n!} \cdot \frac{(n-i)!}{(n-k-i+1)!}\right) \\ &= \frac{k(n-k)!}{n!} \sum_{i=1}^{n} i \frac{(n-i)!}{(n-k-i+1)!} \\ &= \frac{n+1}{k+1} \end{aligned}$$

The worst case running time is $n - k + 1$.

i. Which of the three searching algorithms would you use? Explain your answer.

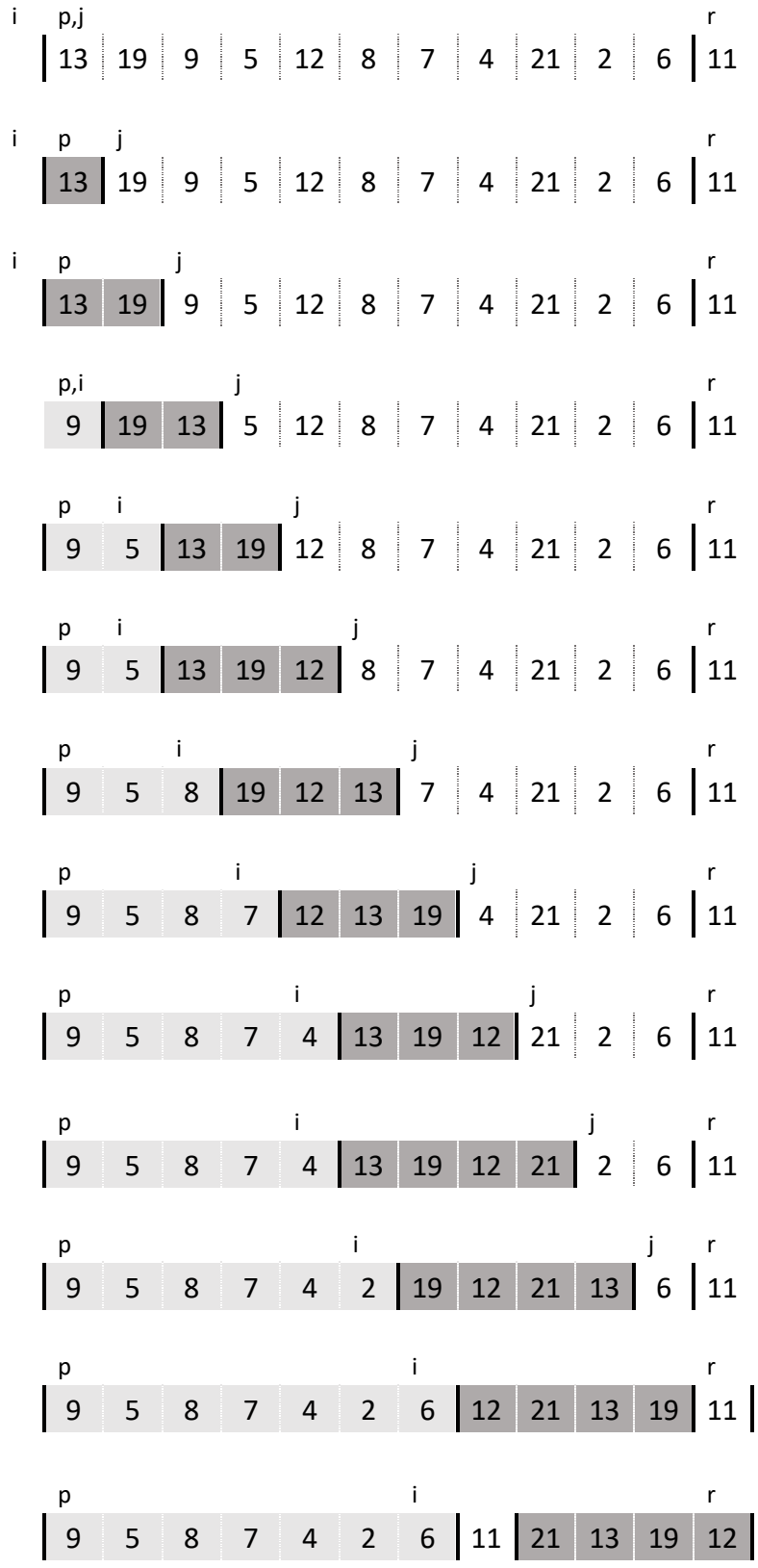
	$k = 0$		$k = 1$		$k > 1$	
	avg	**worst**	**avg**	**worst**	**avg**	**worst**
RANDOM-SEARCH	$n \ln n$	∞	n	∞	$\frac{n}{k}$	∞
DETERMINISTIC-SEARCH	n	n	$\frac{n+1}{2}$	n	**?**	$n-k+1$
SCRAMBLE-SEARCH	n	n	$\frac{n+1}{2}$	n	$\frac{n+1}{k+1}$	$n-k+1$

I would not use RANDOM-SEARCH because of its impractical worst-case running time. Its average running times are also inferior to the other two algorithms.

DETERMINISTIC-SEARCH and SCRAMBLE-SEARCH have very similar running times. However, by taking a random permutation, the latter ensures that the input has a uniform distribution indeed, so on average we will get the calculated expected running time. DETERMINISTIC-SEARCH cannot enforce this distribution, and it might have poor performance for skewed/biased inputs.

As a result, I would choose SCRAMBLE-SEARCH.

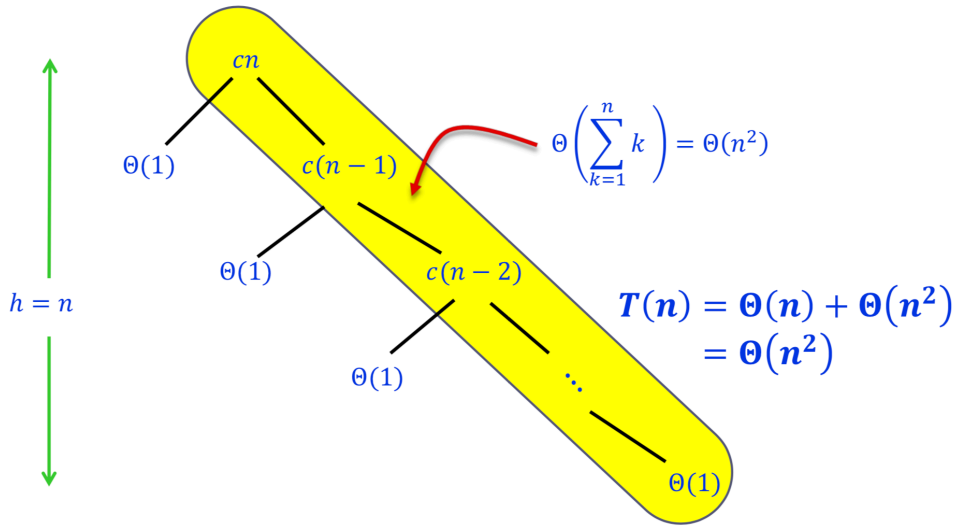
CLRS 7.1-1. Illustrate the operation of PARTITION on the array $A = \langle 13, 19, 9, 5, 12, 8, 7, 4, 21, 2, 6, 11 \rangle$. (pg. 173)



CLRS 7.2-2. What is the running time of QUICKSORT when all elements of array A have the same value? (pg. 178)

If all elements of the input array A have the same value a, in each step, all the remaining $r - p$ elements will end up on one side of the pivot after partitioning. (I.e. either all will be on its left side or all on its right side, depending on how we define PARTITION.)

This in turn means that in every step we can only decrease the input size by 1. This corresponds to the worst-case recursion tree on slide 6/29.



The corresponding worst-case running time is $\Theta(n^2)$.