ADw5

Name : **Peter Garamvoelgyi** Student ID: **2018280070**

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

**Input**: A sequence ofnumbers and a value

**Output**: An indexs.t.or the special value nil ifdoes not appear in.

|  |  |
| --- | --- |
| Deterministic-Search:  1 **for**  **to**  2 **if** **then**  3 **return**  4 **return** nil | Scramble-Search:  1Permute  2 **return** Deterministic-Search |

**g. Suppose that there are no indicessuch that. 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 anoperation, we get

**h. Lettingbe the number of indicessuch that, give the worst-case and expected running times of Scramble-Search for the cases in whichand. Generalize your solution to handle the case in which.**

If, we have the same situation as in g.: The average-case execution coincide with the worst-case execution, the running time isin both cases.

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

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

Ifrepresents the number of comparisons we need to make, then

From this we can directly calculate the expected value

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

For, let us introduce another indicator random variable representing the number of comparisons we need to make:

To have, we first have to

1. Pickelements that are not:
2. Pick one of the’s:
3. Pick a permutation of the remaining elements:
4. Divide byto avoid counting identical permutations multiple times.

(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.)

Letbe a random variable corresponding to the number of indices we must pick.

Using the linearity of expected value, we get

The worst case running time is.

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

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | |  | |  | |
|  | **avg** | **worst** | **avg** | **worst** | **avg** | **worst** |
| Random-Search |  |  |  |  |  |  |
| Deterministic-Search |  |  |  |  | **?** |  |
| Scramble-Search |  |  |  |  |  |  |

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.**

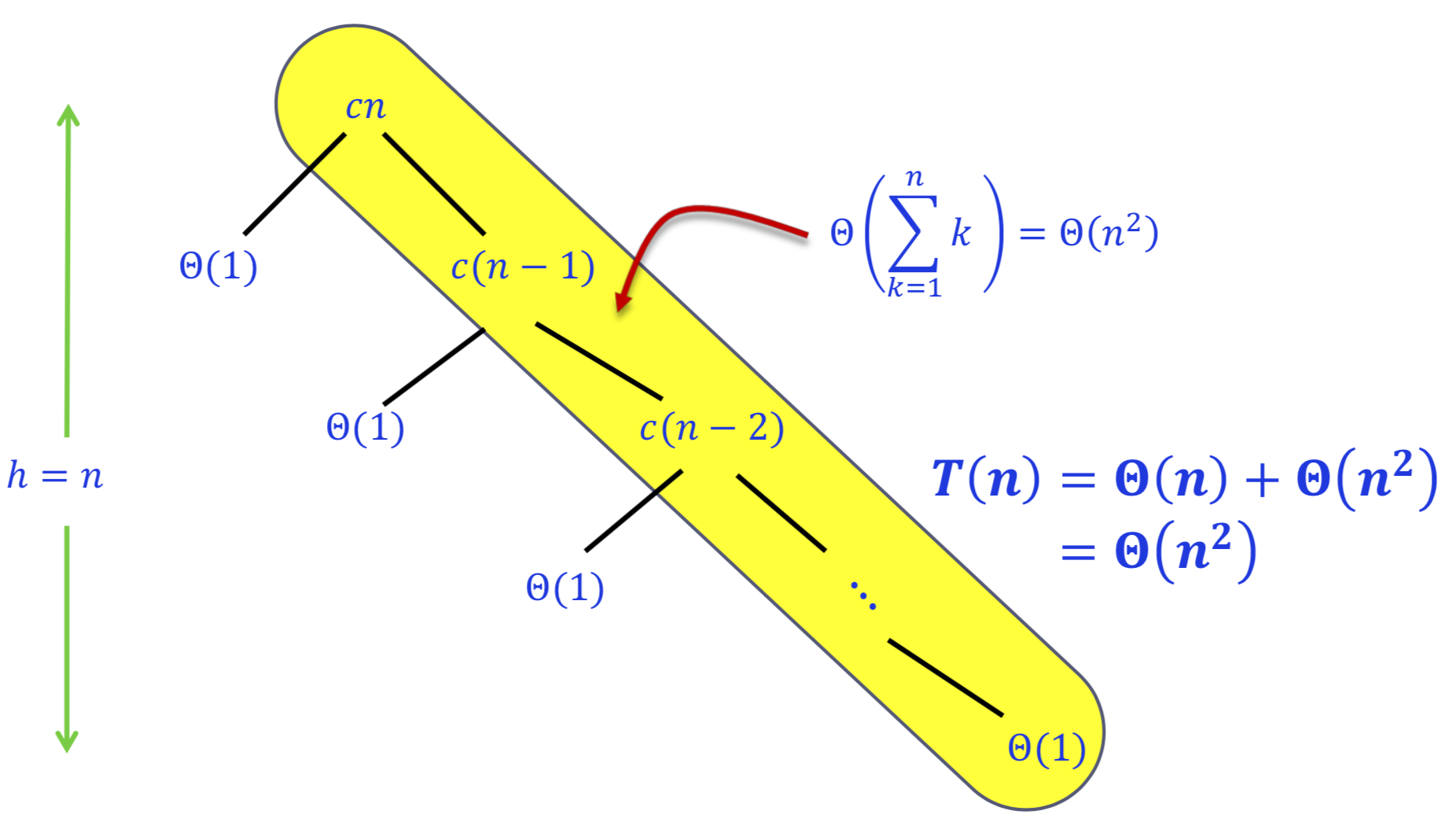
**CLRS 7.1-1. Illustrate the operation of Partition on the array. (pg. 173)**

|  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| i | p,j |  |  |  |  |  |  |  |  |  |  | r |
|  | 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| i | p | j |  |  |  |  |  |  |  |  |  | r |
|  | 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
| i | p |  | j |  |  |  |  |  |  |  |  | r |
|  | 13 | 19 | 9 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p,i |  |  | j |  |  |  |  |  |  |  | r |
|  | 9 | 19 | 13 | 5 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p | i |  |  | j |  |  |  |  |  |  | r |
|  | 9 | 5 | 13 | 19 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p | i |  |  |  | j |  |  |  |  |  | r |
|  | 9 | 5 | 13 | 19 | 12 | 8 | 7 | 4 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p |  | i |  |  |  | j |  |  |  |  | r |
|  | 9 | 5 | 8 | 19 | 12 | 13 | 7 | 4 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p |  |  | i |  |  |  | j |  |  |  | r |
|  | 9 | 5 | 8 | 7 | 12 | 13 | 19 | 4 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p |  |  |  | i |  |  |  | j |  |  | r |
|  | 9 | 5 | 8 | 7 | 4 | 13 | 19 | 12 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p |  |  |  | i |  |  |  |  | j |  | r |
|  | 9 | 5 | 8 | 7 | 4 | 13 | 19 | 12 | 21 | 2 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p |  |  |  |  | i |  |  |  |  | j | r |
|  | 9 | 5 | 8 | 7 | 4 | 2 | 19 | 12 | 21 | 13 | 6 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p |  |  |  |  |  | i |  |  |  |  | r |
|  | 9 | 5 | 8 | 7 | 4 | 2 | 6 | 12 | 21 | 13 | 19 | 11 |
|  |  |  |  |  |  |  |  |  |  |  |  |  |
|  | p |  |  |  |  |  | i |  |  |  |  | r |
|  | 9 | 5 | 8 | 7 | 4 | 2 | 6 | 11 | 21 | 13 | 19 | 12 |

**CLRS 7.2-2. What is the running time of Quicksort when all elements of arrayhave the same value? (pg. 178)**

If all elements of the input arrayhave the same value, in each step, all the remainingelements 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.**