

Homework #6

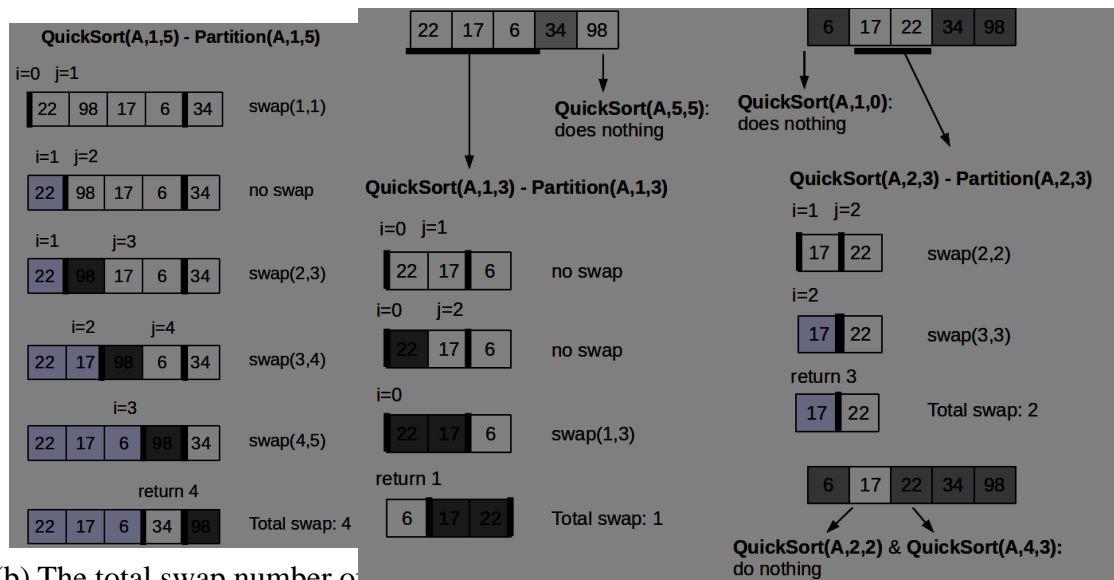
1. (20 points) **QuickSort:** array $A = \langle 22, 98, 17, 6, 34 \rangle$.

(a) (10 points) **Illustrate the operation of QuickSort** using Fig. 7.1 on p. 172 as a **model for the operation of Partition**.

(b) (10 points) **How many swaps are performed by QuickSort to sort array A? How does this compare with the number of swaps used by HeapSort for this same array.**

Ans:

(a) Firstly, apply $\text{QuickSort}(A, 1, 5)$ on A, there $\text{Partition}(A, 1, 5)$ performs 4 swaps and return 4. Then, continue to apply QuickSort on the partitions, $\text{QuickSort}(A, 1, 3)$ will execute $\text{Partition}(A, 1, 3)$ which performs 1 swap and return 1, and $\text{QuickSort}(A, 5, 5)$ will do nothing. After that, $\text{QuickSort}(A, 1, 0)$ and $\text{QuickSort}(A, 2, 3)$ will be executed. $\text{QuickSort}(A, 2, 3)$ will execute $\text{Partition}(A, 2, 3)$ which takes 2 swaps and return 3, while $\text{QuickSort}(A, 1, 0)$ does nothing. The $\text{QuickSort}(A, 1, 5)$ will finish after $\text{QuickSort}(A, 2, 2)$ and $\text{QuickSort}(A, 4, 3)$ finished with no effect.



(b) The total swap number of QuickSort(A, 1, 5) is $4 + 1 + 2 = 7$. Compared to HeapSort on A which has 8 swaps, the performance of QuickSort is better than that of HeapSort.

2. (20 points) QuickSort Analysis: Textbook Exercise 7.4-2 on p. 184.

Ans:

Method1:

The best-case time for procedure QUICKSORT on an input of size n is:

$$T(n) = \min_{0 \leq q \leq n-1} (T(q) + T(n-q-1)) + \theta(n)$$

We guess $T(n) \geq cn \lg n$ for some constant c , we obtain:

$$T(n) \geq \min_{0 \leq q \leq n-1} (cq \lg q + c(n-q-1) \lg(n-q-1)) + \theta(n)$$

Let $F(q) = cq \lg q + c(n-q-1) \lg(n-q-1)$, we have:

$$\begin{aligned} F'(q) &= \lg q + q \frac{1}{q \ln 2} + [\lg(n-q-1) + (n-q-1) \frac{1}{(n-q-1) \ln 2}](-1) \\ &= \lg q - \lg(n-q-1) \end{aligned}$$

Let $F'(q)=0$, we know that $F(q)$ will get the minimum value when $q=(n-1)/2$. Therefore,

$$\begin{aligned} T(n) &\geq c \cdot F\left(\frac{n-1}{2}\right) + \theta(n) \\ &= c \left[\frac{n-1}{2} \lg \frac{n-1}{2} + \frac{n-1}{2} \lg \frac{n-1}{2} \right] + \theta(n) \\ &= cn \lg \frac{n-1}{2} - c \lg \frac{n-1}{2} + \theta(n) \\ &= cn \lg(n-1) - c(n \lg 2 + \lg \frac{n-1}{2}) + \theta(n) \\ &\geq cn \lg \left(\frac{n}{2}\right) - c(n \lg 2 + \lg \frac{n-1}{2}) + \theta(n) \quad (\text{since } n-1 \geq \frac{n}{2} \forall n \geq 2) \\ &\geq cn \lg n - c((2 \lg 2)n + \lg \frac{n-1}{2}) + \theta(n) \\ &\geq cn \lg n \end{aligned}$$

since we can pick the constant c small enough so that the term $c(2n \lg 2 + \lg \frac{n-1}{2})$ will be dominated by the $\theta(n)$ term. Thus, $T(n) = \Omega(n)$.

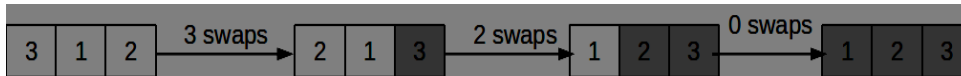
3. (20 points) Suppose that the **PARTITION** procedure of **QUICKSORT** is modified to always use the first element as the pivot:

(a) Provide a worst-case example for **QUICKSORT**.

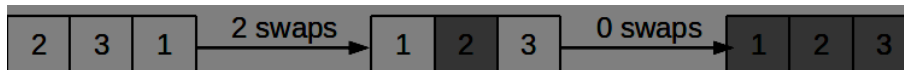
(b) Provide a best-case example for **QUICKSORT**.

Ans:

(a) Worst-case: $\langle 3, 1, 2 \rangle$



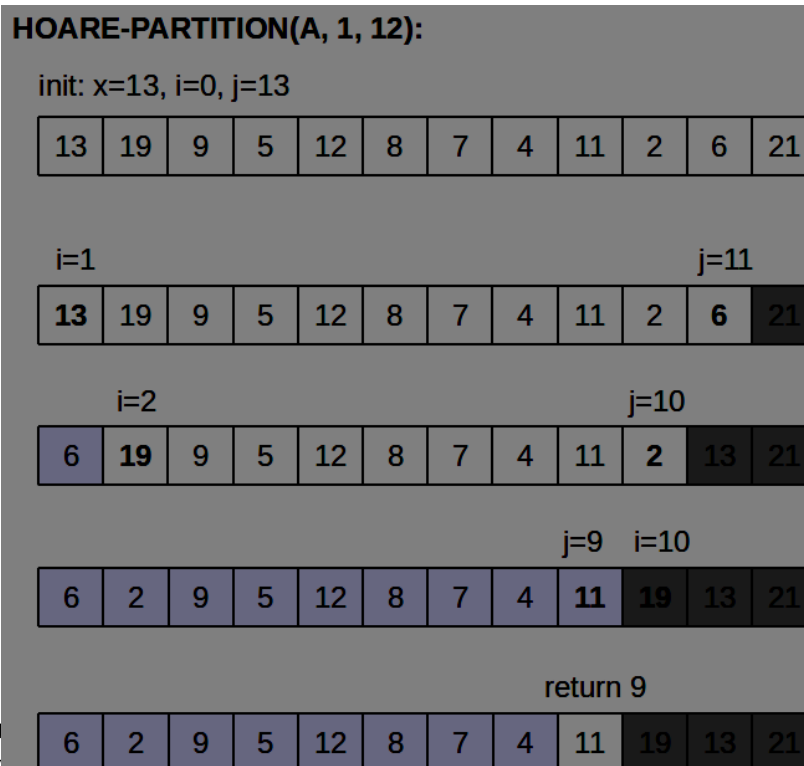
(b) Best-case: $\langle 2, 3, 1 \rangle$



4. (40 points) **Hoare Partition:** Textbook Prob. 7-1 on p. 185-186, parts(a)-(d)

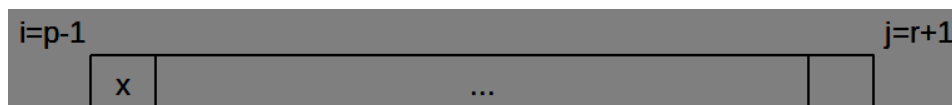
Ans:

(a) Here, we only show the illustration of **HOARE-PARTITION(A,1,12)**, the procedure has 3 iterations of the while loop and breaks the loop at $i=10$ and $j=9$, finally returns 9:

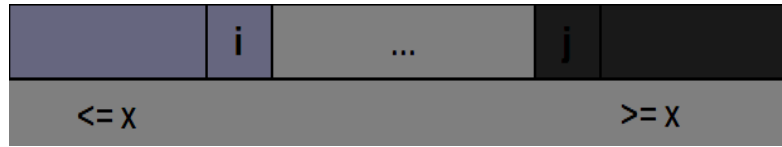


(b) Assume $r > p$ since $A[p..r]$ contains...

The initial state of the array should be as follows:



According to the code, we know that j decreases from $j=r+1$, and this decreasing at least terminates at $j=p$ since $A[p]=x$. Similarly, the increasing of i terminates after increasing 1. After the initial loop, the starting state of the array in any loop can be of the following form:



Note that the light-shadowed part is the smaller part in where all elements are smaller or equal to x , correspondingly the heavy-shadowed part is the larger part in where all elements are greater or equal to x . This is satisfied because of line 5-10.

Therefore, for each iteration of the while loop, it guarantees that the decreasing of j (line 5-7) will stop at the rightmost element of the smaller part even if the all elements in the middle are greater than x . Similarly, i cannot exceed the leftmost element of the larger part. And we know that the size of each parts is at least 1 since $r > p$, so we can conclude that the decreasing j can not be smaller than p , and the increasing i can not be greater than r for $r > p$.

(c) We already prove that $j \geq p$ in problem (b). Now we only need to prove $j < r$:

We find j will decrease if $A[j] > x$, there is two cases in the first loop:

- 1) If $A[j=r] > x$: j will decrease which means $j < r$.
- 2) If $A[j=r] \leq x$: j will not decrease in the first iteration, then $j=r$. Due to line 8-10, i will stop at $i=p$ since $A[p]=x \leq x$, then $i=p$. Therefore, we know that $i=p < r=j$. According to line 11-13, the while loop will continue, which means in the second iteration j will decrease at least 1 anyway. So, j becomes smaller than r in the second iteration.

(d) As shown in (b), when the procedure terminates there are only two parts of the array: the smaller part ($A[k], k=p, \dots, i$) and the larger part ($A[k], k=j, \dots, r$), where $i \geq j$ (due to line 11-13). We know that elements in larger part should be greater or equal to x and elements in smaller part should be smaller or equal to x , that is, elements of $A[p..j]$ (subset of the smaller part) are smaller or equal to elements of $A[j+1..r]$ (subset of the larger part).