Jade Yu Cheng
ICS 311
Homework 5
Sep 11, 2008

Question for lecture 6

Problem 7-4 on p. 162

Stack depth for quicksort
The QUICKSORT Algorithm of Section 7.1 contains two recursive calls to itself. After the call to PARTITION, the left subarray is recursively sorted and then the right subarray is recursively sorted. The second recursive call in QUICKSORT is not really necessary; it can be avoided using an iterative control structure. This technique, called *tail recursion*, is provided automatically by good compilers. Consider the following version of QUICKSORT, which simulates tail recursion.

```
QUICKSORT'(A, p, r) {
  while(p < r) {
    int q = PARTITION(A, p, r);
    QUICKSORT'(A, p, q - 1);
    p = q + 1;
  }
}
```

a. Argue that QUICKSORT' (A, 1, *length*[A]) correctly sorts the array A.

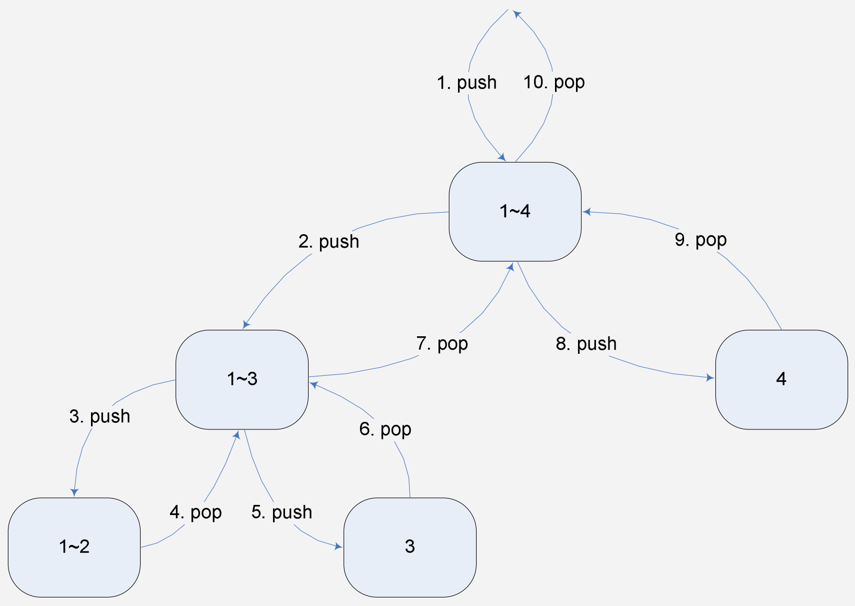
Answer: The book proved that QUICKSORT correctly sorts the array A. QUICKSORT' differs from QUICKSORT in only the last line of the loop. It is clear that the conditions starting the second iteration of the while loop in QUICKSORT' are identical to the conditions starting the second recursive call in QUICKSORT. Therefore, QUICKSORT' effectively performs the sort in the same manner as QUICKSORT. Therefore, QUICKSORT' must correctly sort the array A.

```
QUICKSORT (A, p, r) {
  while(p < r) {
    int q = PARTITION(A, p, r);
    QUICKSORT(A, p, q - 1);
    QUICKSORT(A, q + 1, r);
  }
}
```

Compilers usually execute recursive procedures by using a *stack* that contains pertinent information, including the parameter values, for each recursive call. The information for the most recent call is at the top of the stack, and the information for the initial call is at the bottom. When a procedure is invoked, its information is pushed onto the stack; when it terminates, its information is popped. Since we assume that array parameters are represented by pointers, the information for each procedure call on the stack requires $O(1)$ stack space. The stack depth is the maximum amount of stack space used at any time during a computation.

b. Describe a scenario in which the stack depth of QUICKSORT' IS $\Theta(n)$ on an n-element input array.

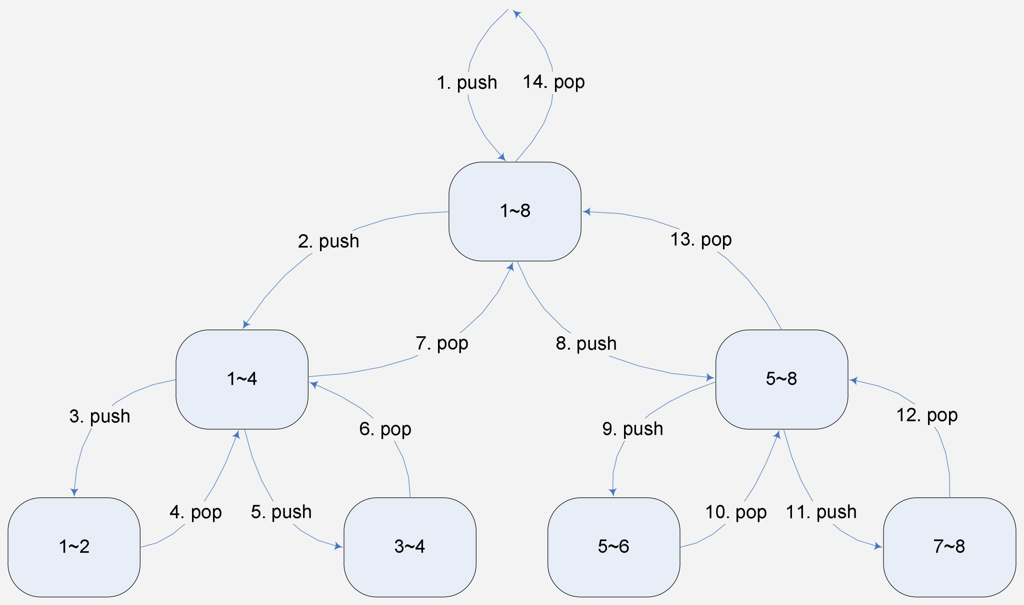
Answer: If partition operations happen on their worst case every time, the stack depth would be $\Theta(n)$, as shown on the diagram.



The stack depth is reached after the third recursive function call which is $n-1$, $\Theta(n)$.

c. Modify the code for QUICKSORT' so that the worst-case stack depth is $\Theta(\lg n)$. Maintain the $O(n \lg n)$ expected running time of the algorithm.

Answer: First of all, if the partition operations happen with the best case, the recursive tree above would look like the following:



The runtime is $O(n \lg n)$, and in the example, the stack depth is reached after the third recursive function call, which is $\Theta(\lg n)$.

As proved in the book, the expected runtime for the average case is $O(n \lg n)$. The recursion tree would look something like the diagram above. Therefore the expected stack depth is also $\Theta(\lg n)$. But it doesn't promise runtime $O(n \lg n)$, nor stack depth $\Theta(\lg n)$.

Solution 1—If random partitions are used then the overall performance would be the average case in QUICKSORT'. In this case, runtime is always $O(n \lg n)$, and the stack depth is always $\Theta(\lg n)$.

Solution 2—Another modification is to add a comparison operation before each recursive call and to perform the partition operations always on the smaller subarrays. This way, the stack depth is promised to be $\Theta(\lg n)$. The overall runtime, however, is increased by $c \lg n$.

$$O(n \lg n) + O(\lg n) = O(n \lg n)$$

Either of these solutions satisfies the problem. The algorithm has a stack depth of $\Theta(\lg n)$, and the expected running time is $O(n \lg n)$.