

## CSE3358 Problem Set 4

02/04/05

Due 02/11/05

### Problem 1: S0rt1ng

(a) (10 points) Consider an array  $A$  of  $n$  numbers each of which is either **0** or **1**. We will refer to such an array as *binary*. Describe an asymptotically efficient algorithm for sorting a binary array  $A$  of size  $n$ . For full credit you should:

- Describe your algorithm in english
- Provide a clear, neat, and complete pseudocode for your algorithm similar to the way we do it in class
- Analyze correctly the running time of your algorithm and express the running time in either a  $\Theta$  or an  $O$  notation.
- Achieve a running time that is the best possible asymptotically

(b) (10 points) We define a Lomuto Quicksort algorithm as a Quicksort algorithm that (recursively) partitions an array into a left sub-array with all elements  $\leq$  Pivot, and a right sub-array with all elements  $>$  Pivot.

Argue (your argument must be very clear, organized, and convincing, if not a formal proof) that any Lomuto Quicksort algorithm will have an average running time of  $\Theta(n^2)$  on binary arrays of size  $n$  (i.e. even when pivot is chosen randomly). Explain why this does not contradict the result that QUICKSORT has an average running time of  $\Theta(n \log n)$ .

### Problem 2: Variations of lists

(1 point each) For each of the following types of lists, what is the asymptotic ( $\Theta$  notation) worst-case running time of the following dynamic-set operations? (make sure you understand what each operation does).

	unsorted singly linked	sorted singly linked	unsorted doubly linked	sorted doubly linked
search(L,k)				
insert(L,x)				
delete(L,x)				
successor(L,x)				
predecessor(L,x)				
minimum(L)				
maximum(L)				

### Problem 3: Recursion and Stacks

(a) (5 points) Implement in C or C++ the following Quicksort algorithm using the Hoare PARTITION (Hoare is the guy who invented Quicksort).

```
PARTITION(A,p,r)
x←A[p]    ▷ this is the pivot
i←p-1    ▷ i starts at the far left
j←r+1    ▷ j starts at the far right

while TRUE
  do repeat j←j-1    ▷ move j down until it finds an element ≤ x
    until A[j]≤ x
  repeat i←i+1    ▷ move i up until it finds an element ≥ x
    until A[i]≥ x
  if i<j        ▷ if they did not cross
    then swap A[i]↔A[j]
    else return j
```

```
QUICKSORT(A,p,r)
if p < r
  then q ←PARTITION(A,p,r)
    QUICKSORT(A,p,q)    ▷ note here the difference from the one we saw
    QUICKSORT(A,q+1,r) ▷ in class which uses a Lomuto PARTITION
```

(b) (5 points) Implement in C or C++ a doubly linked list. Each element of the doubly linked list should be able to hold some data. For instance you can use the following declaration:

```
struct Element {
  Element * prev;
  Element * next;
  void * data; //anything you need for the specific application
};
```

(c) (5 points) Using the linked list of part (b), implement a stack that keeps track of its depth. The stack should provide the following functionality:

- s.push(x): pushes an Element x onto the stack
- s.pop(): pops an Element off the stack and returns it
- s.depth(): returns the number of Elements in the stack
- s.max(): returns the maximum depth ever attained by the stack

(d) (20 points) Compilers usually execute recursive procedures by using a stack that contains parameter values for each recursive call. The information for the most recent call is at the top of the stack, and the information of 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.

Give a non-recursive implementation in C or C++ of the above Quicksort algorithm using your stack of part (c).

Obtain the maximum stack depth for several examples including:

- large sorted array
- large array with all elements equal
- large reverse sorted array

and report the maximum depth of the stack ( as returned by `s.max()` ) as well as the running time in  $\Theta$  notation.

	max. stack depth	running time
sorted array		
array with all elements equal		
reverse sorted array		

(e) (5 points) Can you see that the depth of the stack is  $\Theta(n)$  in the worst-case? Which scenarios lead to such a worst-case?

The stack depth that you explicitly observed in part (d) is the stack depth that the compiler uses for recursion. A deep stack means more internal stack operations and more memory usage. It is possible to modify the QUICKSORT code to make the stack depth always  $\Theta(\log n)$  even when the worst case occurs, without affecting its average case performance of  $\Theta(n \log n)$ . We can do this by first reducing the number of recursive calls using a technique called tail recursion. When the recursive call is the last instruction in the function it can be eliminated. Here's how:

```

QUICKSORT( $A, p, r$ )
while  $p < r$ 
    do  $q \leftarrow$  PARTITION( $A, p, r$ )
        QUICKSORT( $A, p, q$ )
         $p \leftarrow q + 1$ 

```

(f) (5 points) Argue that this new code for QUICKSORT still correctly sorts the array  $A$ .

(g) (5 points) Describe a scenario where the depth of the internal stack is still  $\Theta(n)$  even with the above modification.

(h) (5 points) How can you further modify the code to make the stack depth  $\Theta(\log n)$  without affecting the average case performance of QUICKSORT? *Hint:* since we use recursion on one part of the array only, why not use it on the smaller part?

#### Problem 4: Building a Heap

Consider the problem of building a heap from an array  $A$ . As we saw in class, this can be done by repeated calls to HEAPIFY.

```
BUILD-HEAP( $A$ )  
  for  $i \leftarrow \lfloor n/2 \rfloor$  downto 1  
    do HEAPIFY( $A, i$ )
```

- (a) (5 points) Suggest another *inplace* algorithm for building a heap and provide its pseudocode and analyze its running time.
- (b) (5 points) Does your algorithm produce the same result as the one based on repeated HEAPIFY? If yes, provide an argument. If no, provide a counter example.
- (c) (5 points) Given  $n$  distinct elements, how many different heaps of these elements can you build (I have no idea now)? How does this number change with  $n$  asymptotically?