COMP 340 **Analysis of Algorithms**

Problem Set 2

Assigned: February 7, 2020 *Due: February 21, 2020*

Always provide explanations and show as much work as possible. Designing algorithms often involves some creativity, so start early and work consistently. If you are stuck on a problem, move on and come back to it. If you get stuck again, discuss it with your classmates and/or come see me in office hours.

1. A few years ago I was writing a special *event logger*. The idea was that my program would run alongside a user's program, recording a particular type of event. Whenever the event occurred, my program would save some details about it, and once the user's program ended my program would write all the event details to a log[1]. In other words, I needed some kind of container that allowed inserting at the end and traversing through the elements in the order they were inserted. Importantly, nothing else was required, including search and delete.

   After a few prototypes, I came to the conclusion that both plain linked lists and dynamic arrays were too slow. A linked list wasted too much space on pointers, required too many memory allocations[2], and was too slow to iterate through since the items were not contiguous in memory. A dynamic array spent too much time copying elements during resizing[3].

   (a) Design a hybrid of these two options with the following properties:
      - $O(1)$ **worst-case** `push` (i.e. insert at end)
      - Simple $O(n)$ traversal through all the elements
      - No copying of elements ever occurs
      - Large groups of elements are stored contiguously
      - Only $O(\log n)$ memory allocations

   (b) Briefly explain why your design satisfies the above properties.

   (c) Say I wanted to access some particular index. How would I do this and how long would it take in the worst case, assuming the above data structure with n items.

2. 3-6.

3. 3-12. Notice that the black box only returns *true* or *false*; we want to find the actual subset S.

4. 4-1.

5. 4-5.

6. 4-8.

7. 4-12. *Hint*: Read subsection 4.3.4. For bonus, do this in time $O(n \log k)$ using only $O(k)$ extra space (aside from the input array).

8. 4-14.

[1]For technical reasons my program could not simply write the event information to a file at each individual event; it had to collect them into a batch.

[2]In practice, memory allocation can be expensive

[3]Even though some data structures and algorithms are efficient in terms of Big-O, sometimes the constants that Big-O hides are quite large and are important in practice.

9. 4-17.

10. 4-20.

11. 4-33.

12. Find the closed form for the following recurrence relations by using the recursion tree method. In other words, draw the recursion tree, determine the height of the tree, compute the cost at each level, and sum the cost across the levels. Assume $O(1)$ work and span at the base case, i.e. at the leaves.

    (a) $T(n) = 3T(n/2) + n$
    (b) $T(n) = 2T(\sqrt{n}) + \log n$
    (c) $T(n) = \max(T(n/3), T(n/4)) + \log n$
    (d) $T(n, m) = T(n/2, m/4) + n^2 m$

13. Find the closed form for the following recurrence relations using the Master theorem.

    (a) $W(n) = 3W(n/2) + n$
    (b) $S(n) = 2S(n/4) + n$
    (c) $W(n) = 2W(n/4) + n^2$
    (d) $W(n) = 2W(n/2) + n \log n$

14. The following questions involve the "Stock market" problem, which comes from a parallel algorithms course at WUSTL[4]:

    > The problem with the stock market is that, while it is possible to make a great deal of money buying and selling stocks, it's easy to lose even more. The long-standing—if somewhat unhelpful—maxim to make more money than you lose is "buy low, sell high."
    > The stock market problem is finding the best opportunity to follow this advice: for any sequence of integer prices, where the index in the sequence represents time, find maximum jump from an earlier price to a later price. For example, if the sequence of prices was
    > $$\langle 40, 20, 0, 0, 0, 1, 3, 3, 0, 0, 9, 21 \rangle$$
    > then the maximum jump is 21, which happens between the price at time 2 and time 11. More formally, the stock market problem is to compute
    > $$\max\{s_j - s_i \mid 0 \le i \le j < |s|\}$$
    > Note that this maximum is only well defined if there is at least one element in s.

    (a) Give a brute force algorithm for the Stock market problem. Analyze the runtime.
    (b) Give a divide-and-conquer algorithm for this problem.
    (c) Analyze the runtime of your divide and conquer algorithm.
    (d) Prove your divide and conquer algorithm is correct.

15. (Bonus) 3-15.

[4]Washington University in St. Louis