Introduction to Computer Science (COL100) Minor II March 25, 2019

Name: ARPIT SAXENA **Entry:** 2018MT10742 **Grp:** 29

Note: Maximum marks : 60 (a mark a minute). However, in principle, you can score 85/60 in this exam. You also have the option to submit only the unattempted questions on Moodle, along with a declaration of originality (see http://www.cse.iitd.ac.in/~suban/COL100/#HonourCode), by 1800 hrs. The Moodle submissions will be graded with 50% weightage. The maximum possible marks for the evening submission will be capped at 30. All notations are standard (as done in class). You can present your algorithms either in ML or in Python syntax. Answer only in the space provided.

---

1. Derive the complexity of binary search on a sorted (linked/SML) list. (10 marks)

We'll use the following implementation of binary search in SML:

```
fun split (a, n) = (* assert : n = length of a)
  let
     fun help([], i) = ([], []) (* assert: i = no. of elements in left array)
       | help(x::xs, i) =
           if i=0 then ([], x::xs)
           else
             let
                val (l1, l2) = help(xs, i-1)
             in
                (x::l1, l2)
             end;
  in
     help(a, n div 2)
  end;

fun bsearch([], x, n) = false
  | bsearch(a, x, n) =    (* assert: n = length of a)
    let
       val (l1, y::l2) = split(a, n)
    in
       if y=x then true
       else if y>x then bsearch(l1, x, n div 2)
       else bsearch(l2, x, n-ndiv2-1)
    end;
```

First, we'll evaluate the time complexity of split:

split (a, n) calls help (a, n div 2)

We observe that help tail recursively calls it self n div 2 times,

thus, we can say split runs in n div 2 time.

Now, we find the time complexity of bsearch.

We notice that the worst case time complexity of bsearch would be when the element we're looking for is smaller than the smallest element of array. For that, we have the following recurrence:

$$T(n) = \begin{cases} 0 & \text{if } n = 0 \\ (n \text{ div } 2) + T(n \text{ div } 2) & \text{otherwise} \end{cases}$$

Let k be the smallest integer s.t. $2^k > n \Rightarrow n \text{ div } 2^k = 0$

Then, $T(n) = (n \text{ div } 2) + \cancel{T(n \text{ div } 2)}$ (Using $(n \text{ div } 2^i) \text{ div } 2 = n \text{ div } 2^{i+1}$)

$\cancel{T(n \text{ div } 2)} = n \text{ div } 2^2 + \cancel{T(n \text{ div } 2^2)}$

$\cancel{T(n \text{ div } 2^{k-1})} = n \text{ div } 2^k + \cancel{T(n \text{ div } 2^k)}$

$\cancel{T(n \text{ div } 2^k)} = 0$

$\therefore T(n) = n \text{ div } 2 + n \text{ div } 2^2 + \cdots + n \text{ div } 2^k$

$< n/2 + n/2^2 + \cdots + n/2^k$

$< n\left(\frac{1}{2} + \frac{1}{2^2} + \cdots + \frac{1}{2^k}\right) < n\left(\frac{1}{2} + \frac{1}{2^2} + \cdots\right)$

$< n$

$\Rightarrow T(n) = O(n)$

$\therefore$ Time complexity of binary search in SML list is $O(n)$

4. Consider the problem of finding the maximum value of $\sum_{k=i}^{j} a_k$ (maximum subsequence sum) of integers $a_0, a_1, \ldots, a_{n-1}$. By convention the maximum subsequence sum is 0 if all the integers are negative. For example, for input $-2, 11, -4, 13, -5, -2$, the answer is 20 (a_1 through a_3). Assume that the input sequence is available as an array. Give as efficient an algorithm as possible. (20 marks)

```
def max-sum(a, n):
    #assert: a is an array, ~~...~~ n ≤ len(a)

    i, s1, s2 = 0, 0, 0

    #INV: s1 is the max. sum of all subsequences in a[0.. i-2]
    # and s2 is the max. sum of all subsequences of a[0..i-1] which
    #include a[i-1], 0≤i≤n
    while i < n:
        s1 = max(s1, s2)
        s2 = max(s2 + a[i], 0 + a[i])
        i = i+1

    s = max(s1, s2)

    #assert: s is the maximum ~~seq~~ subsequence sum of a[0..n-1]
    return s
```