1) Master Theorem

Let x(n) be an eventually non decreasing function that satisfies the recurrence relation:

$T(n) = a \cdot T(\frac{n}{b}) + f(n)$; $a = b^k$ $(k = 1, 2, 3 \ldots)$

$T(1) = c$ where $a \geq 1$, $b \geq 2$, $c > 0$

if $f(n) \in \Theta(n^d)$ where $d \geq 0$, then

$$T(n) \in \begin{cases} \Theta(n^d) & \text{if } a < b^d \\ \Theta(n^d \cdot \log n) & \text{if } a = b^d \text{ for all } n \\ \Theta(n^{\log_b a}) & \text{if } a > b^d \end{cases}$$

CSE 321 - Homework 2

Due date: **13**/11/2022, 23:59

1. ***20 pts.*** Solve the following recurrence relations by using Master Theorem and give Θ bound for each of them. If any of the relations cannot be solved by using Master Theorem, state that this is indeed the case together with the explanation of the reason. $T(n) = a \cdot T(\frac{n}{b}) + \underbrace{\Theta(n^k \cdot \log^p n)}_{f(n)}$ with $a \geq 1$, $b > 1$, $k \geq 0$, $p \in R$

   (a) $T(n) = 2 \cdot T(\frac{n}{4}) + \sqrt{nlogn}$ → $a = 2, b = 4, k = p = \frac{1}{2}$ $a = b^k$ and $p > -1$
   So, $T(n) = \Theta(n^{\log_4 2} \cdot \log^{(\frac{1}{2}+1)} n) \Rightarrow T(n) = \Theta(n^{\frac{1}{2}} \cdot \log^{\frac{3}{2}} n)$

   (b) $T(n) = 9 \cdot T(\frac{n}{3}) + 5n^2$ → $a = 9, b = 3, k = 2, p = 0$ $a = b^k$ and $p > -1$
   So $T(n) = \Theta(n^{\log_3 9} \cdot \log^{0+1} n)$
   $T(n) = \Theta(n^2 \log n)$

   (c) $T(n) = \frac{1}{2} \cdot T(\frac{n}{2}) + n$ → $a = \frac{1}{2}$ is not ≥ 1 so master theorem cannot be applied.

   (d) $T(n) = 5 \cdot T(\frac{n}{2}) + logn$ → $a = 5, b = 2, k = 0, p = 1$
   $a > b^k$ so $T(n) = \Theta(n^{\log_2 5})$

   (e) $T(n) = 4^n \cdot T(\frac{n}{5}) + 1$ → Exponentials doesn't fit with master theorem so cannot be solved.

   (f) $T(n) = 7 \cdot T(\frac{n}{4}) + nlogn$ → $a = 7, b = 4, k = 1, p = 1$
   $a > b^k$ so $T(n) = \Theta(n^{\log_4 7})$

   (g) $T(n) = 2 \cdot T(\frac{n}{3}) + \frac{1}{n}$ → $a = 2, b = 3, k = -1$
   k is not ≥ 0 so cannot be solved with master theorem.

   (h) $T(n) = \frac{2}{5} \cdot T(\frac{n}{5}) + n^5$ → $a = \frac{2}{5}, b = 5, k = 5, p = 0$ $a \ngeq 1$
   So, cannot be solved using master theorem.

BAŞ Abdülbari Örnekler

2. ***10 pts.*** Apply the insertion sort algorithm to the following array in ascending order. Explain every step in detail. What is the reasoning behind each operation? What is the updated version of the array?

   Ascending

   $A = \{3, 6, 2, 1, 4, 5\}$

```
int i,j, element
for (i=1 ; i<size; i++){ → Start from 2nd element of loop; Continue until last element.
    element = A[i] → Saving element in ith index to variable so when array order changes, it will stay the same.
    j=i-1 → Every time starting j with (i-1) so
    while(j>=0 && A[j]> element){ → so we can compare with previous elements while it's bigger than element.
        A[j+1]=A[j]; → if comparison holds true, move element to the left.
        j--;
    }
    A[j+1]=element; → Put element to it's right place.
}
```

3. ***20 pts.*** Consider an array and a linked list, both with n elements. Answer the following questions for both data structures.

   (a) Analyze the worst-case time complexity of the following operations. Explain your answer in detail.

      i. Accessing the first element. → Linked list holds first element. So no iteration needed. $O(1)$
      Array elements can be accessed through index so $O(1)$

      ii. Accessing the last element. → if linked list doesn't hold its tail as pointer, we need to iterate through all. So it's $O(n)$
      → Array elements are indexed, doesn't matter where it's $O(1)$.

      iii. Accessing any element in the middle. → Even element is in the middle, accessing will be iterating $\frac{n}{2}$ times for linked list. So $O(\frac{n}{2}) = O(n)$
      → Array elements are indexed, doesn't matter where it's $O(1)$.

      iv. Adding a new element at the beginning. → for linked list, we need to change the new element's pointer and connect to old head. $O(1)$
      → for array, we need to create a new array ~~&~~ move every element so it will take $O(n)$

      v. Adding a new element at the end. → for linked list we need to iterate until the end and then add. it's $O(n)$ time
      → for array, if no extra space is allocated, we need to create a new array, move all, and then add. So $O(n)$

      vi. Adding a new element in the middle. → for linked list, we need to iterate half of list so $O(\frac{n}{2}) \Rightarrow O(n)$
      → for array, we need to create a new array, move elements until middle, then add element and then rest of it. So it'll take $O(n)$.

      vii. Deleting the first element. → for linked list, moving head to second element and deallocating the first is enough. So, $O(1)$
      → For array, creating a new array and copying from one to other will take $O(n)$ time.

      viii. Deleting the last element. → for linked list, moving iterator till the end, then deallocating is enough. It's $O(n)$.
      → For array, we can just ignore last element. But for an array without last element, we need to move every element except last. $O(n)$

      ix. Deleting any element in the middle. Both similar to viii, because of iteration over elements, it will take $O(n)$.
      $O(n)$

   (b) Analyze the space requirements.

   → If space requirements mean extra space needed,
   i – No need extra space
   ii – Linked list, we need to create an iterator size of pointer. Array nothing needed.
   iii – Linkedlist needs an iterator to create. For array, nothing needed.
   iv – Linked list, nothing needed. Array, we need to create an array size of n.
   v – Linked list, one element size; array, n element is needed.
   vi – Linked list, one element size; array, n element is needed.
   vii – LL, nothing needed; array, n element is needed.
   viii – LL, nothing needed; array, n element is needed.
   ix: LL, nothing needed; array, n element is needed.

4 - if root is None :
return

4. ***15 pts.*** Construct an algorithm that converts a given binary tree with size n to a binary search tree (BST). Make sure you preserve the structure of the tree, i.e. you should not add or delete a node. Write down the pseudo-code of the algorithm, explain your reasoning, and analyze the best-case, worst-case, and average-case time complexities.

5. ***15 pts.*** Consider an integer array $A = \{a_0, a_1, ..., a_n\}$ and an integer x. You are asked to find a pair (a_i, a_j), if any, within this array such that $|a_i - a_j| = x$. Design an algorithm with $O(n)$ time complexity to solve the problem. Write down the pseudo-code of the algorithm, and explain your reasoning in detail.

6. ***20 pts.*** For each of the statements below, indicate true or false with the explanation of the reason (explain the reason for each of them, not only for false ones).

   (a) Shape of a BST (full, balanced, etc.) depends on the insertion order.

   (b) The time complexity of accessing an element of a BST might be linear in some cases.

   (c) Finding an array's maximum or minimum element can be done in constant time.

   (d) The worst-case time complexity of binary search on a linked list is $O(log(n))$ where n is the length of the list.

   (e) Worst-case time complexity of the insertion sort algorithm is $O(n)$ if the given array is reversely sorted.