Algorithm Design and Analysis (Fall 2023)

Assignment 1

Deadline: Nov 1, 2023

1. (25 points) Prove the following generalization of the master theorem. Given constants $a \geq 1, b > 1, d \geq 0$, and $w \geq 0$, if $T(n) = 1$ for $n < b$ and $T(n) = aT(n/b) + n^d \log^w n$, we have

$$T(n) = \begin{cases} O(n^d \log^w n) & \text{if } a < b^d \\ O(n^{\log_b a}) & \text{if } a > b^d \\ O(n^d \log^{w+1} n) & \text{if } a = b^d \end{cases}.$$

Solution:

1. $a < b^d$:

   I will prove this case by induction on n. Knowing that $T(n) \leq aT(\frac{n}{b}) + cn^d \log^w n$ for some constant c, we claim that $T(n) \leq Bn^d \log^w n$ holds for some constant B s.t. $(1 - \frac{a}{b^d})B > c$.

   Base step:

   $T(b) \leq a + cb^d \log^w b \leq Bb^d \log^w b$ holds trivial.

   Inductive step:

   Assuming $T(i) \leq Bi^d \log^w i$ holds for $i = 1, ..., n-1$. For $i = n$,

$$\begin{aligned} T(n) &\leq aT(\frac{n}{b}) + cn^d \log^w n \\ &\leq aB(\frac{n}{b})^d \log^w \frac{n}{b} + cn^d \log^w n. \end{aligned}$$

   If $aB(\frac{n}{b})^d \log^w \frac{n}{b} + cn^d \log^w n \leq Bn^d \log^w n$, then we are done. Let $n = b^x$, then we need to prove

$$aB(b^d)^{x-1}(x-1)^w \log^w b \leq (B-c)(b^d)^x x^w \log^w,$$

   which can be reduced to

$$\frac{a}{b^d}B \leq (\frac{x}{x-1})^w B - c.$$

   When $n \to \infty$, $x \to \infty$, $\frac{x}{x-1} \to 1^+$, we only need to prove

$$\frac{a}{b^d}B \leq B - c,$$

   which goes trivial under the hypothesis $(1 - \frac{a}{b^d})B > c$.

2. $a > b^d$:

   According to the basic master theorem, for $T(n) = aT(\frac{n}{b}) + O(n^d)$, if $a > b^d$, then $T(n) = O(n^{\log_b a})$, which is independent of d. We can always find $\epsilon \to 0$ such that $a > b^{d+\epsilon}$, then for $T(n) = aT(\frac{n}{b}) + O(n^{d+\epsilon})$, $T(n) = O(n^{\log_b a})$ still holds.

   It is obvious that $n^d < n^d \log^w n < n^{d+\epsilon}$ holds for any ϵ when $n \to \infty$. Let $\epsilon \to 0$, then we have $T(n) = O(n^{log_b a})$ if $a > b^d$ for general case $T(n) = aT(\frac{n}{b}) + O(n^d \log^w n)$.

3. $a = b^d$:

   I will prove this case by induction on n extremely similar to case 1. Knowing that $T(n) \leq aT(\frac{n}{b}) + cn^d \log^w n$ for some constant c, we claim that $T(n) \leq Bn^d \log^{w+1} n$ holds for some constant B s.t. $B > a + c$.

   Base step:

   $T(b) \leq a + cb^d \log^w b \leq Bb^d \log^w b$ holds trivial.

   Inductive step:

   Assuming $T(i) \leq Bi^d \log^{w+1} i$ holds for $i = 1, ..., n-1$. For $i = n$,

$$\begin{aligned} T(n) &\leq aT(\frac{n}{b}) + cn^d \log^w n \\ &\leq aB(\frac{n}{b})^d \log^{w+1} \frac{n}{b} + cn^d \log^w n. \\ &= Bn^d \log^{w+1} \frac{n}{b} + cn^d \log^w n \end{aligned}$$

   If $Bn^d \log^{w+1} \frac{n}{b} + cn^d \log^w n \leq Bn^d \log^{w+1} n$, i.e. $B \log^{w+1} \frac{n}{b} + c \log^w n \leq B \log^{w+1} n$, then we are done.

   When $n \to \infty$, since $c \log^w n$ has has a lower power exponent, this term can be reasonably ignored. $B \log^{w+1} \frac{n}{b} \leq B \log^{w+1} n$ goes trivial.

2. (25 points) Recall the median-of-the-medians algorithm we learned in the lecture. It groups the numbers by 5. What happens if we group them by $3, 7, 9, \ldots$? Please analyze those different choices and discuss which one is the best.

Solution:

The time complexity $T(n)$ of the median-of-the-medians algorithm mainly consists of three parts: calculating medians of each group at the cost of cn, finding the median of medians by running an independent select at the cost of $T(an)$, and the actual select recursion at the cost of $T(bn)$. Therefore, it can be written as

$$T(n) = T(an) + T(bn) + cn.$$

Assuming we group numbers by g, I will analyse these three terms.

1. $T(an)$:

   Grouping numbers by g reduces the size of the list of medians to $\frac{n}{g}$, and this is exactly the list which we need to run an independent select algorithm on. So $T(an) = T(\frac{n}{g})$.

2. $T(bn)$:

   Since g is an odd, we can make sure that there are $\frac{g+1}{2g}$ numbers in each $\frac{1}{2} \cdot \frac{n}{g}$ groups less or greater than the pivot. Therefore, $T(bn) = T(n - \frac{g+1}{2g} \cdot \frac{1}{2} \cdot \frac{n}{g}) = T(\frac{3g-1}{4g}n)$.

3. cn:

   To calculating medians of each g numbers, we consider the simplest bubble sorting method. We only need to perform $\frac{g+1}{2}$ bubbles to find the median, and for ith bubble, $g - i$ comparisons are needed. The total number of the comparisons is

$$\begin{aligned} cn &= \frac{n}{g} \sum_{i=1}^{\frac{g+1}{2}} g - i \\ &= \frac{3(g-1)(g+1)}{8g} n. \end{aligned}$$

According to the above analysis, we have

$$T(n) = T(\frac{3g+3}{4g}n) + \frac{3(g-1)(g+1)}{8g}n.$$

If $g = 3$, then $\frac{3g+3}{4g} = n$, thus this still leaves n numbers to search in, not reducing the problem sufficiently.

For $g > 5$, $T(n) = O(n)$ according to the master theorem. Let $T(n) = \alpha n$, the constant α matters. We can get

$$\alpha n = \frac{3g+3}{4g}\alpha n + \frac{3(g-1)(g+1)}{8g}n$$
$$\alpha = \frac{3(g+1)(g-1)}{2g-3}.$$

α decreases as g increases when $g > 5$. So $g = 5$ is the best choice.

Reference:

Median of medians - Wikipedia

3. (25 points) Let X and Y be two sets of integers. Write $X \succ Y$ if $x \geq y$ for all $x \in X$ and $y \in Y$. Given a set of m integers, design an $O(m \log(m/n))$ time algorithm that partition these m integers to k groups $X_1, \ldots, X_k$ such that $X_i \succ X_j$ for any $i > j$ and $|X_1|, \ldots, |X_k| \leq n$. Notice that k is not specified as an input; you can decide the number of the groups in the partition, as long as the partition satisfies the given conditions. You need to show that your algorithm runs in $O(m \log(m/n))$ time.

   Remark: We have not formally define the asymptotic notation for multi-variable functions in the class. For f and g be functions that maps $\mathbb{R}^k_{>0}$ to $\mathbb{R}_{>0}$, we say $f(\mathbf{x}) = O(g(\mathbf{x}))$ if there exist constants $M, C > 0$ such that $f(\mathbf{x}) \leq C \cdot g(\mathbf{x})$ for all $\mathbf{x}$ with $x_i \geq M$ for some i. The most rigorously running time should be written as $O(m \cdot \max\{\log(m/n), 1\})$, although it is commonly just written as $O(m \log(m/n))$ for this kind of scenarios.

   Solution:

   The algorithm is below. Function *FindPivot* is just part of median-of-medians algorithm to find the median.

Algorithm 1

Require: original set X, capacity limit n, function *FindPivot*

Output: the output set of k sets $\mathcal{S}$

function *Partition*(set A)

1: **if** $|A| \leq n$ **then**
2: append A to $\mathcal{S}$
3: **return**
4: **end if**
5: pivot = *FindPivot*(A)
6: Divide A into two sets, A_1 contains all elements less than pivot, and A_2 is opposite.
7: *Partition*(A_1), *Partition*(A_2)

end function

   The time complexity of this algorithm (recorded briefly as $\mathcal{O}$) is equivalent to the difference of two parts: applying algorithm without truncation (doesn't terminate when $|A| \leq n$, record briefly as $\mathcal{A}$) on original set X, and applying $\mathcal{A}$ on k output subsets $X_1, ..., X_k$. To prove $\mathcal{O}$ is $O(m \log \frac{m}{n})$, we only need to prove the $\mathcal{A}$ is $O(m \log m)$ since

$$T_{\mathcal{O}}(m) \leq Bm \log m - k \cdot Bn \log n \leq Bm \log m - \frac{m}{n} \cdot Bn \log n = Bm \log \frac{m}{n}.$$

   Since choosing the pivot using *FindPivot* is $O(m)$, and we can ensure at least $\frac{3}{10}$ elements are less than it, so $T_{\mathcal{A}}(m) = T_{\mathcal{A}}(km) + T_{\mathcal{A}}((1-k)m) + O(m)$ s.t. $k \geq 0.3$. I'll prove $T_{\mathcal{A}}(m) = O(m \log m) \leq Bm \log m$ by induction on m.

$$\begin{aligned} T_{\mathcal{A}}(m) &\leq Bkm \log km + B(1-k)m \log(1-k)m + cm \\ &= Bm \log m + Bm(k \log k + (1-k) \log(1-k)) + cm \\ &\leq Bm \log m \end{aligned}$$

where $B(k \log k + (1-k) \log(1-k)) + c \leq 0$. Then we are done.

4. (25 points) Given an array $A[1, \ldots, n]$ of integers sorted in ascending order, design an algorithm to **decide** if there exists an index i such that $A[i] = i$ for each of the following scenarios. Your algorithm only needs to decide the existence of i; you do not need to find it if it exists.

   (a) The n integers are positive and distinct.

   (b) The n integers are distinct.

   (c) The n integers are positive.

   (d) The n integers are positive and are less than or equal to n.

   (e) No further information is known for the n integers.

   Prove the correctness of your algorithms. For each part, try to design the algorithm with running time as low as possible.

Solution:

(a) The algorithm for this part is below. The time complexity is $O(1)$.

To prove the correctness, if $A[1] = 1$, we find $i = 1$. Else, $A[1] > 1$, which is the opposite to $A[1] = 1$ since integers are positive. For any $i \in [1, n]$, there must be $A[i] > A[1] + (i - 1) = i$ since integers are distinct.

Algorithm 2

Input: $A[1, ..., n]$, n sorted integers are positive and distinct

Output: $True$ or $False$

```
1: if A[1] = 1 then
2:     return True
3: else
4:     return False
5: end if
```

(b) The algorithm for this part is below. The time complexity is $O(\log n)$.

In this algorithm, we use binary search to check the intermediate element, and there are three possible scenarios here.

1. If $A[mid] = mid$, then we find such i.
2. If $A[mid] < mid$, there can't be an i s.t. $i < mid$ and $A[i] = i$, otherwise $A[mid] \geq A[i] + (mid - i) = mid$ since elements are distinct. Therefore, i may only exist in the right of mid within the search scope.

3. Similarly, if $A[mid] > mid$, i may only exist in the left of mid within the search scope for the same reason.

Algorithm 3

Input: $A[1, ..., n]$, n sorted integers are distinct

Output: $True$ or $False$

```
initialization: left = 1, right = n
while left + 1 ≤ right do
    mid = (left + right)//2
    if A[mid] = mid then
        return True
    else if A[mid] < mid then
        left = mid + 1
    else
        right = mid − 1
    end if
end while
return False
```

(c) The algorithm for this part is below. The worst-case time complexity of this algorithm is $O(n)$, but it is better than the naive traversal algorithm.

Algorithm 4

Input: $A[1, ..., n]$, n sorted integers are positive

Output: $True$ or $False$

```
initialization: i = 1
while A[i] ≤ n do
    if A[i] = i then
        return True
    else
        i = A[i]
    end if
end while
return False
```

This is an optimized linear search, starting from $A[1]$. For each element, there are also three scenarios.

1. If $A[i] = i$, then we find such an i.

2. If $A[i] > i$, since the array is sorted, $A[i+1], A[i+2], ..., A[A[i]-1]$, all of these elements must be equal or greater than $A[i]$, violating $A[k] = k$, and can be passed. We only need to search $A[i]$ next. It is worth pointing out that this trick applies to all scenarios as long as array is sorted.
3. If $A[i] < i$, this scenarios can never occur when we search $A[i]$ using this algorithm otherwise there must exist an k s.t. $k < i$ and $A[k] = k$ (proof is in part(d), which is a generalization of this claim) and we can find such k before we search i.

(d) For this assumption ($A[n]$ is a sorted array in ascending order, $A[1] \geq 1$, $A[n] \leq n$), we claim that there always exists i such that $A[i] = i$. I'll prove this by induction on the size of array n.

Base step:

If $n = 1$, $A[1] = 1$, then we find $i = 1$.

Inductive step:

Assuming the claim holds when $n = k$.

For $n = k + 1$, $A[k + 1] \leq k + 1$, and then we know $A[k] \leq A[k + 1] \leq k + 1$. If $A[k] = k + 1$, then $A[k + 1]$ must be $k + 1$, we find $i = k + 1$. Else, $A[k] \leq k$, according to the assumption, we can find an $i \in [1, k]$. Therefore, claim holds when $n = k + 1$.

(e) The algorithm for this part is below. The time complexity is $O(n)$, but it is also better than the naive traversal algorithm. The proof is same to the second scenarios of case(c).

Algorithm 5

Input: $A[1, ..., n]$, n integers are sorted

Output: $True$ or $False$

```
initialization: i = 1
while i ≤ n do
    if A[i] = i then
        return True
    else if A[i] < i then
        i = i + 1
    else
        i = A[i]
    end if
end while
return False
```

5. How long does it take you to finish the assignment (including thinking and discussion)? Give a score (1,2,3,4,5) to the difficulty. Do you have any collaborators? Please write down their names here.

Solution:

These four questions cost 4h, 4h, 5h, 5h respectively (typing also included).

The difficulty score is 5, 4, 5, 4 respectively.

Collaborators: **Guo Jing** (discuss together on third question), **T.A. Bu** (take inspiration on first question).