Algorithm HW#1

B02901178 江誠敏

November 1, 2016

1 Problem 1.

I think the problem is not so well-defined since for an algorithm, these properties may depend on the implementations (e.g. Which item to choose when merging two sorted list in merge sort if the first element of two lists are equal). We shall assume that the "most common" implementations are used.

	Worst-case time complexity	Stable?	Time complexity on sorted	Time complexity on reversed
Bubble sort	$\Theta(n^2)$	Yes	$\Theta(n^2)$	$\Theta(n^2)$
Merge sort	$\Theta(n \log n)$	Yes	$\Theta(n \log n)$	$\Theta(n \log n)$
Insertion sort	$\Theta(n^2)$	Yes	$\Theta(n)$	$\Theta(n^2)$
Quick sort	$\Theta(n^2)$ [1]	No[2]/Yes[3]	$\Theta(n^2)$	$\Theta(n^2)$
Heap sort	$\Theta(n \log n)$	No	$\Theta(n \log n)$	$\Theta(n \log n)$

2 Problem 2.

Lemma 1 (Binary search)**.** *Let $C = (c_i)$ be an sorted array (i.e. $c_1 \le c_2 \le \cdots \le c_n$). Then the function* `lower_bound`$(x) := \min_{c_i \in C, c_i \ge x} c_i$ *could be compute in $\mathcal{O}(\log n)$.*

Proof. Let $\hat{c} =$ `lower_bound`(x). Consider $m = \lfloor n/2 \rfloor$ and let $L = (c_1, c_2, \cdots, c_m)$, $R = (c_{m+1}, c_{m+2}, \cdots, c_n)$.
If $c_m \ge x$, then $c_m \le \hat{c}$ hence $c_m \in L$. if $c_m < x$, then $c_m < x \le \hat{c} \implies c_m < \hat{c}$, so $\hat{c}$ is in R. Then we could recursive preform the algorithm on L or R since they are also sorted array until there is only one elements in the sequence. Assuming random access model, The time complexity is

$$T(n) = T(\lceil n/2 \rceil) + \mathcal{O}(1) \implies T(n) = \mathcal{O}(\log n)$$

$\square$

[1]Assume that the median of medians method isn't used
[2]If Lomuto's method is used
[3]If Hoare's method is used, which is mentioned in class

Let $b_i := -(a_i + i)$. Notice that we shall not pre-calculate these b_is (or it would cost $\mathcal{O}(n)$), but lazily evaluate them. That is, we shall evaluate b_i only when needed (in the binary search coroutine).

Since $a_i > a_{i+1}$ and a_i, a_{i+1} are integers, so $a_i \geq a_{i+1} + 1$. Hence

$$b_i = -(a_i + i) \leq -a_{i+1} - 1 - i = -(a_{i+1} - (i+1)) = b_{i+1}$$

,thus $b_1 \leq b_2 \leq \cdots \leq b_n$ and we could preform binary search.

Then

$$a_i + i = n \iff -b_i = n \iff b_i = -n.$$

And notice that $\exists i : b_i = -n \iff$ `lower_bound`$(-n) = -n$. By Lemma 1, `lower_bound`$(-n)$ could be calculated in $\mathcal{O}(\log n)$, hence the above described an algorithm that runs in $\mathcal{O}(\log n)$.

3 Problem 3

We shall assume that n, k are powers of 2.

1. Let $m = n/2, h = k/2$. Since $|\{i : 1 \leq i \leq h\}| = h = k/2$, so $\bigcup_{i \leq h} g_i$ are the shorter half and $\bigcup_{i > h} g_i$ are the taller half of the children, since each g_i has the same size and every child in g_i are taller than every child in group g_j for all $i > j$ by definition. Hence

$$\begin{aligned}\bigcup_{i \leq h} g_i &= \{a_i : a_i \text{ is in the smaller half of } A = (a_i)\} \\ &= \{a_i : a_i \leq \hat{a}\}\end{aligned}$$

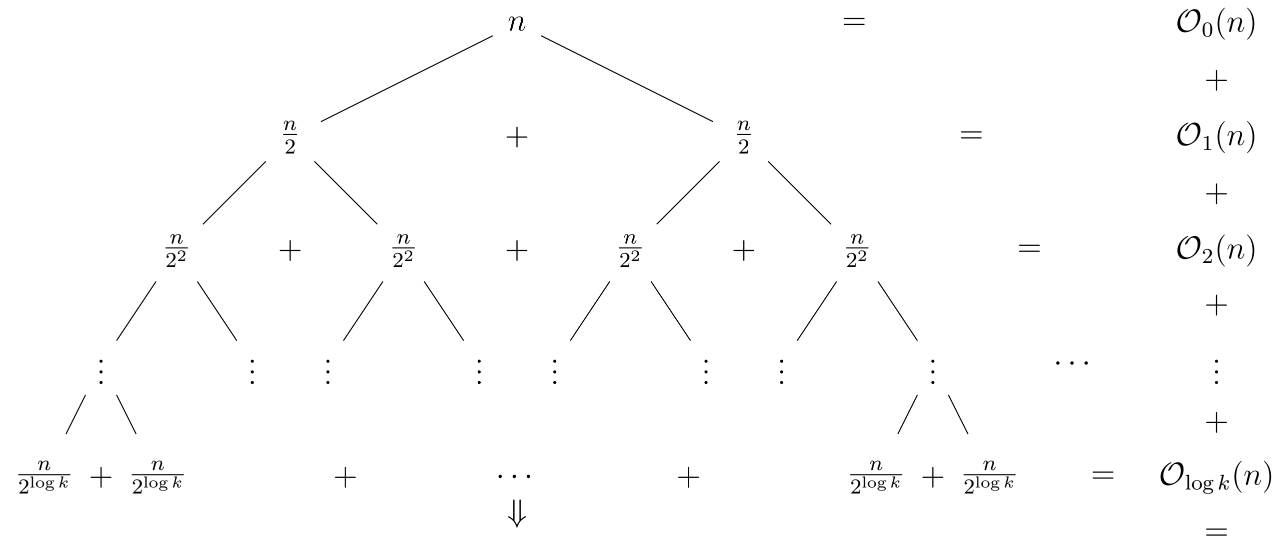
   Where $\hat{a}$ is the h-th smallest element in $A = (a_i)$. Using the order statistic algorithm with medians of medians mention in class, $\hat{a}$ could be calculated in $\mathcal{O}(n)$. Then partition the children into two groups

$$L = \bigcup_{i \leq h} g_i = \{a_i : a_i \leq \hat{a}\}, \quad R = \bigcup_{i > h} g_i = \{a_i : a_i > \hat{a}\}.$$

   which could be done in $\mathcal{O}(n)$ time once we know $\hat{a}$. Then we turn the problem into two subproblems "Spliting L, R into h equal-sized groups g_{Li}, g_{Ri} respectively, satisfying $g_{Li} < g_{Lj}$ and $g_{Ri} < g_{Rj}$ $\forall i > j$" that is same as the original problem, and then $(g_{L1}, g_{L2}, \cdots, g_{Lh}, g_{R1}, g_{R2}, \cdots, g_{Rh})$ would be the solution to the original problem. The time complexity is

$$T(n,k) = 2T(n/2, k/2) + \mathcal{O}(n) \implies T(n,k) = \mathcal{O}(n \log k)$$

Where the complexity is calculated using recursion tree [4].


n = $\mathcal{O}_0(n)$
+
$\frac{n}{2}$ + $\frac{n}{2}$ = $\mathcal{O}_1(n)$
+
$\frac{n}{2^2}$ + $\frac{n}{2^2}$ + $\frac{n}{2^2}$ + $\frac{n}{2^2}$ = $\mathcal{O}_2(n)$
+
$\vdots$
+
$\frac{n}{2^{\log k}}$ + $\frac{n}{2^{\log k}}$ + $\cdots$ + $\frac{n}{2^{\log k}}$ + $\frac{n}{2^{\log k}}$ = $\mathcal{O}_{\log k}(n)$
$\Downarrow$
=

$$\mathcal{O}\left(\sum_{i=0}^{\log k} 2^i \cdot \frac{n}{2^i}\right) = \mathcal{O}\left(\sum_{i=0}^{\log k} n\right) = \mathcal{O}(n \cdot k) \quad \Leftrightarrow \quad \mathcal{O}(n \log k)$$

2. We shall site the Stirling's formula first.

   Lemma 2 (Stirling's formula)**.**

   $$\log_2 n! = n \log_2 n - O(n)$$

   Assume that the algorithm uses m comparisons. There are exactly

   $$P = \frac{n!}{(n/k)!^k}$$

   possibility to group children into k equal sized group. And if every child in group g_i is taller than every child in group g_j for every $i > j$, then g_1 should contains the first k shortest children, g_2 should contains the $k+1$ to $2k$-th shortest children ... So the group each child belongs is decided, and hence exactly one of the P possibilities satisfied the constraint. Thus the algorithm cound be view as a function $f(\alpha_1, \alpha_2, \cdots, \alpha_m)$ which base on the true/false result of comparisons, the algorithm has to decide a answer among p possiblities. The size of the domain of f is 2^m and the size of the image is P, since the size of the domain should be larger than the size of the image,

   $$2^m \geq P \implies m \geq \log n! - k \log(n/k)!$$

   And

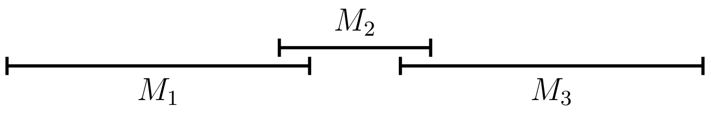
   $$m \geq \log n! - k \log(n/k)! = n \log n - O(n) - k\frac{n}{k}\log(n/k) + kO(n/k)$$
   $$= n \log n - n \log(n/k) \pm O(n) = n \log k \pm O(n) = \mathcal{O}(n \log k)$$

   Hence the time complexity is in $\Omega(n \log k)$.

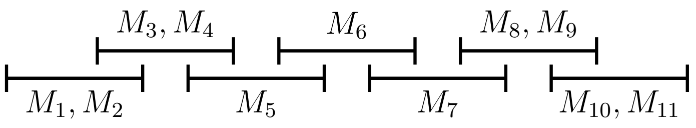
[4]The graph is modified from `http://www.texample.net/tikz/examples/merge-sort-recursion-tree/`

4 Problem 4

1. No. For example let $M_1 = (s_1, t_1) = (0, 10), M_2 = (9, 14), M_3 = (13, 23)$. The length of the meeting is $10, 5, 10$ respectively. The algorithm would choose M_2 first. But then M_2 is conflict with M_1, M_3, so the algorithm stops and give the solution $S = \{M_2\}$. But $\hat{S} = \{M_1, M_3\}$ is a valid solution which is better.



2. No. Let $M_1 = M_2 = (0, 3)$, $M_3 = M_4 = (2, 5)$, $M_5 = (4, 7)$, $M_6 = (6, 9)$, $M_7 = (8, 11)$, $M_8 = M_9 = (10, 13)$, $M_{10} = M_{11} = (12, 15)$. Then every meeting is conflict with 3 other meetings except M_6, which only conflict with 2 mettings. So the algorithm would choose M_6 and remove M_5, M_7. The algorithm would eventually give a solution $S = \{M_\alpha, M_6, M_\beta\}$ where $\alpha \in \{1, 2, 3, 4\}, \beta \in \{8, 9, 10, 11\}$. But $\hat{S} = \{M_1, M_5, M_7, M_{10}\}$ is a better solution.



3. Let $\mathcal{M} = \{M_1, M_2, \cdots, M_n\}$. For a set of meetings S, Define $f(S) = M_i$ such that M_i has the smallest t_i in S.

   Lemma 3. *There is an optimize solution with $f(\mathcal{M})$ chosen.*

   Proof. Let S be an optimal solution. If $f(\mathcal{M}) \in S$ then we're done. Otherwise let $\hat{M} = (\hat{s}, \hat{t}) = f(S)$. Notice that $\forall M = (s, t) \in S, M \neq \hat{M}$ we have $s \geq \hat{t}$, or else by the minimalty of $\hat{M}$ we have $s < \hat{t} \leq t$ and M would conflict with $\hat{M}$. By the minimalty of $(s', t') = f(\mathcal{M})$, we have $t \geq \hat{t} \geq s$, so $f(\mathcal{M})$ would not conflict with any meeting in S except $\hat{M}$. So consider $S' = S \setminus \{\hat{M}\} \cup \{f(\mathcal{M})\}$. Then $|S'| = |S|$ is also an optimal solution with $f(\mathcal{M})$ in it. $\square$

   By the lemma, we could greedily pick the meeting M with earilest ending time t_i and add it to the solution set S. Then we remove all meetings that is conflict with M, and repeat the process until there is no other meetings left. The final solution set we get is an optimal solution. The algorithm could be implement easily to run in $\mathcal{O}(n^2)$ by iterate through meetings and find the one with minimum t_i each time, or even $\mathcal{O}(n \log n)$ if we sort the meetings by ending time.

5 Problem 5

1. Let $f(k, s, c)$ be the maximum number of event that can be held in parallel consider only the events in $A_i = \{a_1, a_2, \cdots, a_i\}$, requires help of no more than s students, and cost no more than c dollars.

   Then $f(0, \bullet, \bullet) = 0$. For $k \geq 1$, we could write the recursive formula of $f(k, s, c)$ by consider whether or not a_k is chosen.

   - If a_k is not chosen, then it is equivalent to consider only the events in A_{k-1}. so $f(k, s, c) = f(k-1, s, c)$.
   - If a_k is chosen, which requires that $s \geq s_i, c \geq c_i$. Then after choosing a_k, we could only choose the events in A_{k-1} and requires no more than $s - s_i$ students and with cost no more than $c - c_i$. So $f(k, s, c) = f(k-1, s - s_i, c - c_i) + 1$.

   Hence

$$f(k, s, c) = \begin{cases} 0 & \text{if } k = 0 \\ f(k-1, s, c) & \text{if } s < s_i \text{ or } c < c_i \\ \max\left(f(k-1, s, c), f(k-1, s - s_i, c - c_i) + 1\right) & \text{Otherwise} \end{cases}$$

   And the solution of the answer is $f(n, n, n^2)$. Notice that the recursive formula is well defined since $f(k, \bullet, \bullet)$ use only values of $f(k-1, \bullet, \bullet)$, and $f(0, \bullet, \bullet) \equiv 0$ is defined, so the recursion would terminate.

   Now, by preforming Top-Down memorization (or Bottom-Up table filling). We could turn the recursion into dynamic programming. The number of states is $|\text{dom } f| = n \cdot n \cdot n^2 = \mathcal{O}(n^4)$. Each of the states depend on at most $2 = \mathcal{O}(1)$ states. So the complexity of dynamic programming is $\mathcal{O}(n^4)\,\mathcal{O}(1) = \mathcal{O}(n^4)$.

2. First we proof some lemmas.

   Lemma 4. *If $G = (V, E)$ is an simple undirected graph and $V = \{v_1, v_2, \cdots, v_n\}$. Let $\pi(v_i) = i$ be the index mapping, and if for all $i \geq 2$, there is exactly one edge (u, v_i) adjacent to v_i satisfied $\pi(u) < i$, then G is a tree.*

   Proof. Let $p(v), v \neq v_i$ be the vertex adjacent to v such that $\pi(p(v)) < \pi(v)$. Then v is connected to $p^k(v)$ for all $k \geq 0$. But since $\pi(p(u)) < \pi(u)$ and $|V|$ finite, there is a k such that $\pi(p^k(v)) = 1 \implies p^k(v) = v_1$ and thus every vertex is connected to v_1.

   Now we shall proof that G doesn't contain loop, or else assume $u_1, u_2, \cdots, u_m$ forms a loop. Since G simple, $m \geq 3$. Let u be the one with smallest π (i.e., $\pi(u) < \pi(u_i), \forall u \neq u_i$). Then u is adjacent with two vertices w each having $\pi(u) < \pi(w)$, which leads to an contradiction.

   Hence G is a connected graph without loop, and thus G is a tree. □

Lemma 5. *If $G = (V, E)$, $V = \{a_1, a_2, \cdots, a_n\}$, and $E = \{(i, \lfloor i/2 \rfloor) : i \geq 2\}$. Then G is a binary tree.*

Proof. Since for all $i \geq 2$, $a_{\lfloor i/2 \rfloor}$ is the only one which is adjacent to i and have index smaller than i, and G is obviously simple. So by Lemma 4, G is a tree.

Notice that for all $x \geq 1$, there are exactly 2 integer with $\lfloor y \rfloor = x$, namely $y = 2x$ and $y = 2x + 1$. So if we pick a_1 as root, each vertex a_i has at most two children, a_{2i} and a_{2i+1}. And each vertex $a_i, i \geq 2$ has a father $a_{\lfloor i/2 \rfloor}$. Hence G is a binary tree. □

Now by Lemma 5, The events a_i and dependencies $(a_i, a_{\lfloor i/2 \rfloor})$ forms a binary tree. Let A_k be the meetings that is in the subtree of a_k. Define $f(k, s, c)$ be the maximum number of event that can be held in parallel consider only the events in A_i, requires help of no more than s students, and cost no more than c dollars. Also define $g(k, s, c)$ to be the same as f except that we specified that a_k could not be chosen. That is, $g(k, s, c)$ is the maximum number of events could be hold with all the constraints satisfied and with a_k not chosen.

If a_k is a leaf, then

$$f(k, s, c) = \begin{cases} 1 & \text{if } s \geq s_i \text{ and } c \geq c_i \\ 0 & \text{otherwise} \end{cases}$$
$$g(k, s, c) = 0$$

If a_k has a single child a_j, then

$$f(k, s, c) = \begin{cases} \max\left(g(j, s, c) + 1, f(j, s, c)\right) & \text{if } s \geq s_i \text{ and } c \geq c_i \\ f(j, s, c) & \text{otherwise} \end{cases}$$
$$g(k, s, c) = f(j, s, c)$$

Since we can't choose a_j if we choose a_k.

If a_k has a two children a_{2k}, a_{2k+1}, then things get complicated. Notice that binary tree didn't have loops, so any event in A_{2k} would not conflict with any event in A_{2k+1}. Hence the optimize solution consider only events in A_{2k} won't conflict with the optimize solution consider only events in A_{2k+1}, excepts that they may conflict with a_k. So we should consider two cases, whether a_k is chosen or not. Also we don't know how many students s', s'' and how much money c', c'' do the optimal solution in events in A_{2k} or A_{2k+1}. So we should enumerate through every possible pair $(s', c'), (s'', c'')$. Hence the recursive formula is

$$f(k,s,c) = \begin{cases} \max\left(\max\limits_{\substack{s'+s''=s-s_i \\ c'+c''=c-c_i}} g(2k,s',c') + g(2k+1,s'',c'') + 1, \right. \\ \left. \qquad \max\limits_{\substack{s'+s''=s \\ c'+c''=c}} f(2k,s',c') + f(2k+1,s'',c'') \right) & \text{if } s \geq s_i \text{ and } c \geq c_i \\ \max\limits_{\substack{s'+s''=s \\ c'+c''=c}} f(2k,s',c') + f(2k+1,s'',c'') & \text{otherwise} \end{cases}$$

$$g(k,s,c) = \max_{\substack{s'+s''=s \\ c'+c''=c}} f(2k,s',c') + f(2k+1,s'',c'')$$

The recursive formula is well defined since $f(k,\bullet,\bullet), g(k,\bullet,\bullet)$ depend on $2k, 2k+1$ only and the recursion eventually reach k' where $a_{k'}$ is a leaf and then terminate. Finally the solution is $f(1,n,n^2)$ since A_1 contains all the events.

Again, by preforming Top-Down memorization (or Bottom-Up table filling). We could turn the recursion into dynamic programming. The number of states is $|\text{dom } f| + |\text{dom } g| = 2n \cdot n \cdot n^2 = \mathcal{O}(n^4)$. Each of the states depend on at most $2 \cdot n \cdot n^2 = \mathcal{O}(n^3)$ states since there are at most n pairs with $s' + s'' = s_i$ and n^2 pairs with $c' + c'' = c_i$. So the complexity of dynamic programming is $\mathcal{O}(n^4)\,\mathcal{O}(n^3) = \mathcal{O}(n^7)$.