HU, Wenbin 20760747 whuak@connect.ust.hk.

Problem 1:

(a) Sore_Algorithm(A[1...n]):

```
1   build a min heap with elements in A[1...k]          //build a heap
2   for i ← k+1 to n:                                   //do insert, extract-min
3       Insert(A[i], k+1)                                  for A[1...k+1], A[2...k+2]...
4       min ← Extract-Min(k+1)                             iteratively
5       Swap(A[i-k], min)
6   j ← k
7   for i ← n-k+1 to n:                                 // sort the last k-th items.
8       A[i] ← Extract-Min(j)
9       j ← j-1
10  return A
```

description: build a heap for k+1 items, and do Insert, Extract_Min iteratively to get smallest, second smallest items..., ect.

(b) proof:

by def of k-Swapped array:

every element in A[k+2 ... n] is greater than A[1]

every element in A[k+3 ... n] is greater than A[2]

⋮

every element in A[k+j ... n] is greater than A[j-1]

∴ the smallest element must be in A[1...k+1]

the 2nd smallest element must be in A[1...k+2]

⋮

the j-th smallest element must be in A[1...k+j]

as for the Alg: we create a heap to get a min priority queue.

in first for: I. we extract min in A[1...k+1], which is the smallest, swap it with A[1]. ∴ A[1] now is smallest

II. insert A[k+2] we extract min in A[2...k+2], which is the 2nd smallest. swap it with A[2]. ∴ A[2] now is 2nd smallest.

⋮

III. insert A[k+j] we extract min in A[j...k+j], which is the j-th smallest. swap it with A[j] ∴ A[j] now is j-th smallest.

∴ iteratively. we $A[1\dots n-k]$ is sorted, and it is $n-k$ smallest elements.

then. we do heap sort for last k elements.

∴ $A[1\dots n]$ is sorted.

(c) line 1 build a min heap use $O(k\log k)$ time.

line 2~5: do $n-k$ times loop:

inside each loop: Insere use $O(\log k)$ time

Extract-Min use $O(\log k)$ time

Swap use $O(1)$ time.

∴ use $O(n-k)\cdot(2\log k+1)$ time in line 2~5

line 7~9:

do k times loop:

inside each loop: Extract-Min use $O(\log k)$ time.

∴ use $O(k\log k)$ time.

∴ $T(n) = O(k\log k) + O((n-k)\cdot(2\log k+1)) + O(k\log k)$

$\le O(k\log k) + O(n\log k)$

$\le O((n-1)\log k) + O(n\log k)$ (since $n \ge k+1$, which makes this problem meaningful)

$= O(n\log k)$

(d) In class, the original array is random, it can be $n!$ possive permutations.

so it must have $\Omega(n\log n)$ running time by comparison alg.

but in this problem. the array is not a random array.

it is a k-swapped array, and it must satisfied $\forall i>j \in [1,n]$, $A[i] < A[j]$, s.t. $i-j \le k$.

to obey its def, this array can't be $n!$ permutations (eg: smallest one is in $A[1\dots k+1]$)

so the outcome of the running is not necessarily $\Omega(n\log n)$

(e) proof:

by the def of k-swapped array, we know the jth smallest element is in $A[1\dots k+j]$

for smallest element. we choose one place in $A[1\dots k+1]$, which has $k+1$ options.

for 2-nd smallest element. choose one place in $[1\dots k+2]$, except the place for smallest, which has $k+1$ options

for j-th smallest element. choose one place in $[1\dots k+j]$, except places for $(j-1)$ smallest which has $k+1$ options

⋮

for $(n-k)$-th element. choose one place in $[1\dots n]$, except those places for $(n-k-1)$ smallest

∴ we have $(k+1)^{n-k}$ options for smallest $(n-k)$ elements.

for last k elements, we have $k!$ options.

$\therefore$ we have $(k+1)^{n-k} \cdot k!$ leaves for the decision tree.

$\therefore$ height: $2^h \geq (k+1)^{n-k} \cdot k!$

$$\Rightarrow h \geq \log_2 (k+1)^{n-k} \cdot k! = (n-k)\log_2(k+1) + \log_2 k!$$

$\therefore$ Running time: $T(n) \geq (n-k)\log_2(k+1) + \log_2 k!$

by String's approximation:

$$\Theta(\log k!) = \Theta(k \log k)$$

$$\therefore T(n) = \Omega\left((n-k)\cdot(\log(k+1)) + k\log k\right)$$

$$\geq \Omega\left((n-k)\log(k+1)\right)$$

$$\geq \Omega((n-k)\log k)$$

$$\frac{(n-k)\log k}{n\log k} = \frac{n-k}{n} = 1 - \frac{k}{n} \geq 1 - \frac{n-1}{n} = \frac{1}{n} \qquad (n \geq k+1)$$

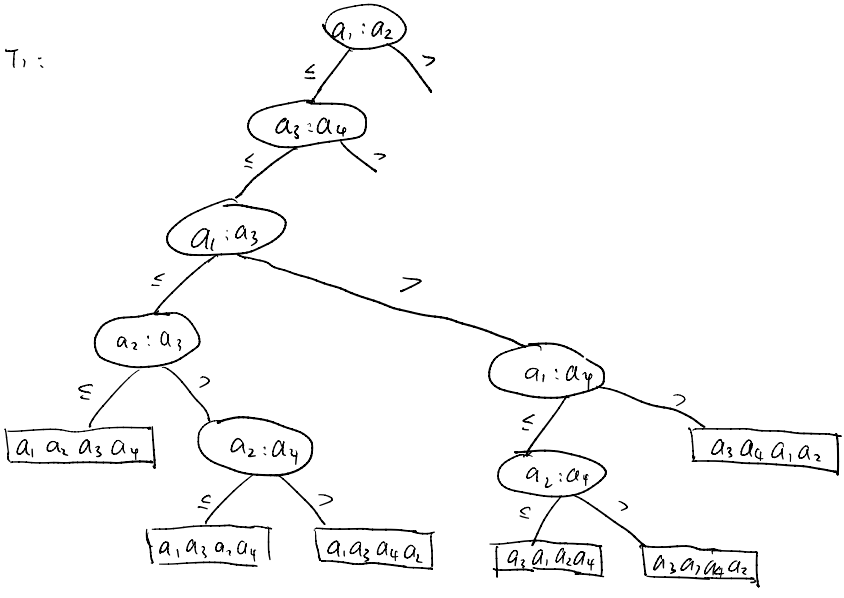
Let $c \leq \frac{1}{n}$

$\therefore \exists n_0 > 0,\ c \leq \frac{1}{n_0}$, $\forall n \geq n_0$, s.t. $T(n) \geq c \cdot (n\log k)$.

$\therefore T(n) = \Omega(n\log k)$.

Problem 2

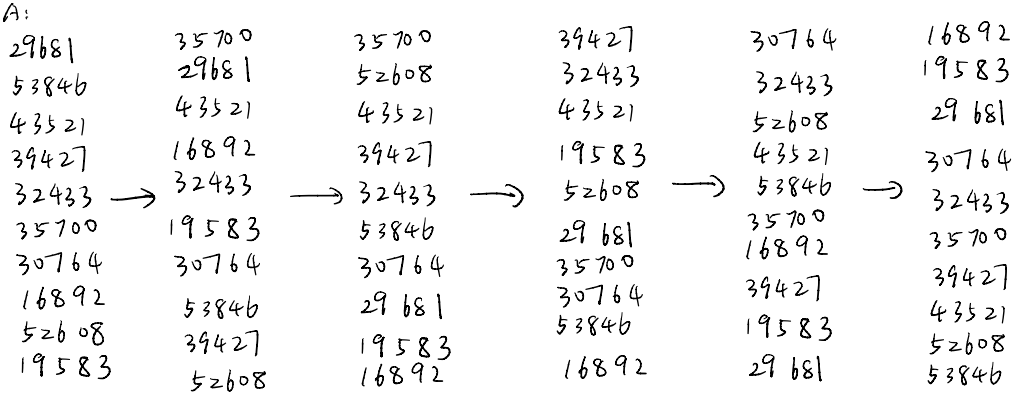
expanding T_1:



Problem 3.

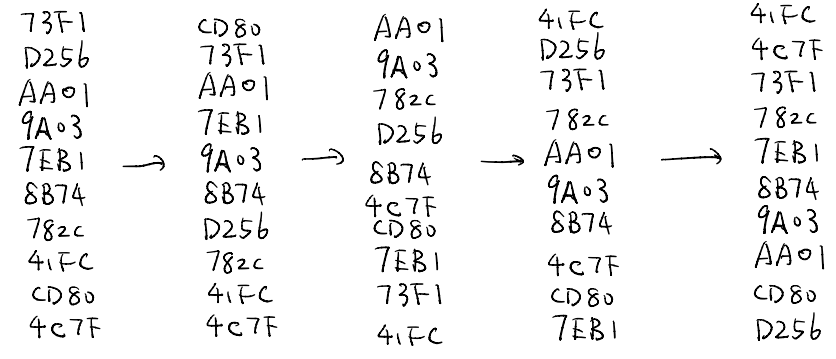
(a)

A:



(b) A = [73F1, D256, AA01, 9A03, 7EB1, 8B74, 782c, 41FC, CD80, 4c7F]

A:



Problem 4:

(a) proof:

let $A = \{f_1, \cdots, f_n\}$. by definition, we know "A cover I" obviously

sort $I_1, I_2 \cdots I_n$ by finishing time. as increasing order

for any I_i: if f_i can not cover any other interval. let Set $S_k = \{f_i\}$

if f_i can cover some other intervals, group them as set S_k.
and remove their f from A

finally, we get a new A.

for every S_k: there is only one point cover interval inside. this point is in A
if we remove this point. those interval won't be covered.

∴ A is a minimal cover
and A is from $\{f_1 \cdots f_n\}$. by removing some items.
∴ $A \subseteq \{f_1 \cdots f_n\}$
∴ ∃ a minimal cover A, s.t. $A \subseteq \{f_1 \cdots f_n\}$

(b) alg:

Sort $I_1, \cdots I_n$ by finishing time as increasing order
create a Set A, last ∈ 0
for i ← 1 to n:
if $s_i \geq$ last, then $A \leftarrow A \cup \{f_i\}$, last ← f_i
return A.

description: do a loop for intervals. once we find a s_i that is bigger than a f(last) we select before, we need the f_i as new point. (because "last" can't cover intervals later)

(c) proof:

Part 1: now we prove A can be a cover of I:
when there is only one interval. f_1 cover I
when there is more than one interval.
assume $I - I_k$ is covered by $A = \{f_1, \cdots, f_m\}$ (inductive hypothesis)
if $s_k \leq \max\{f_1, \cdots, f_m\}$. then I_k is cover by $\{f_1, \cdots f_m\}$
, because $f_k \geq \max\{f_1, \cdots f_m\}$ by greedy alg above, ∴ A covers I.
else. let $A = A \cup \{f_k\}$, then I_k is cover by f_k, which means I is covered by A
∴ By Induction, I is covered by $\{f_1, \cdots f_m, f_k\}$.
∴ proved that A can be a cover of I.

Part II. now prove A is OPT.

Assume Greedy is different from OPT.

let $i_1, i_2 \cdots i_k$ denote the set of points cover I. by Greedy (increasing order)

let $j_1, j_2 \cdots j_m$ denote the set of points cover I by OPT. (increasing order)

chouse largest r such that $i_t = j_t$ for $t \le r$, and $i_{r+1} \neq j_{r+1}$

Create OPT^* from OPT by replace j_{r+1} with i_{r+1} (Greedy step. it is the local optimal)

$\Rightarrow$ OPT^* is also optimal. which shares rel items with Greedy.

$\therefore$ Greedy $\equiv OPT^*$

repeat this process until OPT is the same as greedy.

$\therefore$ proved that my alg is OPT.

d) sort use $O(n \log n)$ time.

create a Set use $O(1)$ time

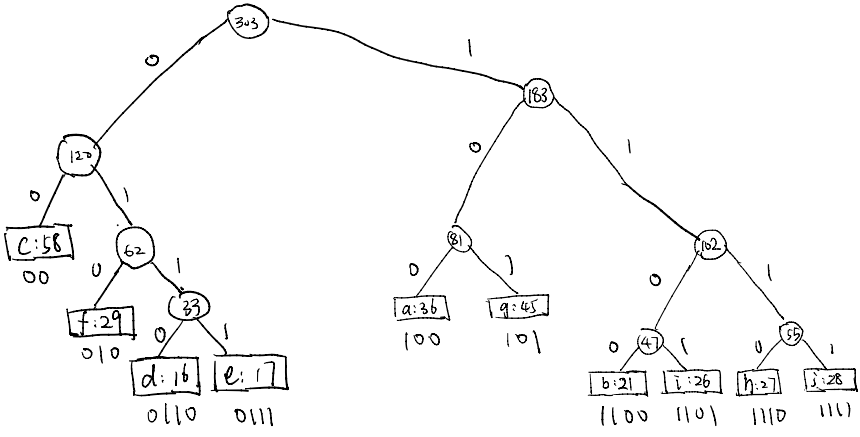
for loop use $O(n)$ time.

$\therefore$ use $O(n \log n)$ time in total.

Problem 5:

i	1	2	3	4	5	6	7	8	9	10
a_i	a	b	c	d	e	f	g	h	i	j
$f(a_i)$	36	21	58	16	17	29	45	27	26	28

(a) Huffman Tree:



(b) codebook:

a	100
b	1100
c	00
d	0110
e	0111
f	010
g	101
h	1110
i	1101
j	1111

(c)

c	00
a	100
f	010
g	101
b	1100
d	0110
e	0111
h	1110
i	1101
j	1111