CSCI 6470
Homework 1
Jiakuan Li

1.

(a) $U(n) = T(n)$ because the worst case takes as long as possible to run. That means it takes most calculation. $U(n)$ is the time upper bound, which is the time to do the most calculation. Therefore, $U(n) = T(n)$.

(b) $U(n) < S(n)$ because $U(n)$ guarantee the most calculation time for any case. $S(n)$ is the best case which used the least calculation.

(c) $L(n) < T(n)$ because $L(n)$ guarantee the least calculation time for any case. $T(n)$ is the case that used the most calculation time.

(d) $L(n) = S(n)$ because $L(n)$ guarantee the least calculation time for any case. $S(n)$ is the best case which used the least calculation.

2.

(a) Big-O notation is referred to as order of the function. It provides an upper bound on the growth rate of the function. It could be Under a constant scaled curve for $f(n)$. For $O(f(n)) + O(f(n))$, if $f(n) = n^2$, $O(f(n)) + O(f(n)) = 2n^2$, which is still $O(n^2)$. In another word, $O(n) + O(m) = MAX(O(n), O(m))$, in this case, $MAX(O(n), O(n)) = O(n)$.

(b) Let $T(n) = 2n, f(n) = n$, $2^{2n} \neq O(n)$.

3.

(a) Prove: $h(n) \leq h(\lfloor \frac{n}{2} \rfloor) + 1$
Assume: $n = 2m + 1$ for all $m > 0$.
LHS:

$$h(n) = 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{2m+1}$$

RHS:

$$\begin{aligned} h(n) &= h(\lfloor \frac{2m+1}{2} \rfloor) + 1 \\ &= h(m) + 1 \\ &= 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{m} + 1 \end{aligned}$$

Then, LHS - RHS, we have

$$\begin{aligned} 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{2m+1} - 1 + \frac{1}{2} + \frac{1}{3} + ... + \frac{1}{m} + 1 &\leq 0 \\ \frac{1}{m+1} + \frac{1}{m+2} + ... + \frac{1}{2m+1} - 1 &\leq 0 \\ \frac{1}{m+1} + \frac{1}{m+2} + ... + \frac{1}{2m+1} &\leq 1 \end{aligned}$$

The LHS of the inequality is $\sum_{k=m+1}^{2m+1} \frac{1}{k}$, so $\frac{1}{m+1}$ is the greatest term of this sequence and it is < 1. The inequality holds and problem proved.

(b) Basecase:
let $n = 4$,

$$h(4) = 1 + h(2) = 1 + 1 + h(1) = 1 + 1 + 1 = 3$$

$$c \log_2 n = 4c$$

the basecase hold for all $c \geq 1$ Assume this hold for $\frac{n}{2}$

$$\begin{aligned} h(n) &\leq h(\lfloor \frac{n}{2} \rfloor) + 1 \\ &\leq c \log_2 \frac{n}{2} + 1 \\ &\leq c(\log_2 n - \log_2 2) + 1 \\ &\leq c \log_2 n - c + 1 \end{aligned}$$

so it is ok for all $c \geq 1$.

4.

(a) Guessed: $cn \log_2 n + 2$
Basecase:
$T(1) = 2 \leq 3n \log_2 n + 2$ holds for all $n \geq 1$
Assume the guessed upper bound holds for $\lfloor \frac{3n}{4} \rfloor$

$$\begin{aligned} T(n) &= 3T(\lfloor \frac{n}{4} \rfloor) + n \\ &\leq 3(c\lfloor \frac{3n}{4} \rfloor \log_2 \lfloor \frac{3n}{4} \rfloor + 2) + n \\ &\leq 3cn \log_2 \lfloor \frac{3n}{4} \rfloor + 6 + n \\ &= 3cn(\log_2 n + \log_2 \frac{3}{4}) + 6 + n \\ &= 3cn \log_2 n + 6 + n + 3cn \log_2 \frac{3}{4} \\ &= 3cn \log_2 n + 2 + 4 + n - 3cn \log_2 \frac{4}{3} \end{aligned}$$

To allow that last term $\leq 3cn \log_2 n + 2$,
We have choose c such that

$$4 + n - 3cn \log_2 \frac{4}{3} \leq 0$$

let $c = \frac{2}{log_2 \frac{4}{3}}$,

$$\begin{aligned}4 + n - 3cn \log_2 \frac{4}{3} &= 4 + n - 3n \frac{2}{\log_2 \frac{4}{3}} \log_2 \frac{4}{3} \\ &= 4 + n - 6n \\ &= 4 - 5n \\ &\leq 0\end{aligned}$$

for all $n \geq 1$. So the upper bound is $O(n \log_2 n)$

(b)

$$\begin{aligned}T(n) &= 3T(\lfloor\frac{n}{4}\rfloor) + n \\ &= 3(3T(\lfloor\frac{n}{4^2}\rfloor + \frac{n}{4}) + n \\ &= 3^2T(\lfloor\frac{n}{4^2}\rfloor) + \frac{3n}{4} + n \\ &= 3^2(3T(\lfloor\frac{n}{4^3}\rfloor) + \frac{n}{4^2}) + \frac{3n}{4} + n \\ &= 3^3T(\lfloor\frac{n}{4^3}\rfloor) + \frac{3n}{4^2} + \frac{3n}{4} + n \\ &= ... \\ &= n(1 + \frac{3}{4} + \frac{3}{4^2} + \frac{3}{4^3} + ... + \frac{3}{4^{i-1}}) + 3^iT(1)\end{aligned}$$

for i iterations, the subproblem has constant time where $\frac{n}{4^i} = 1 \rightarrow i = \log_4 n$. Then we have

$$\begin{aligned}T(n) &= n(1 + \frac{3}{4} + \frac{3}{4^2} + \frac{3}{4^3} + ... + \frac{3}{4^{\log_4 n-1}}) + 3^{\log_4 n}T(1) \\ &= n \sum_{k=0}^{\log_4 n-1} \frac{3}{4^k} + 6^{\log_4 n}\end{aligned}$$

because $n \sum_{k=0}^{\log_4 n-1} \frac{3}{4^k}$ is a geometric sum, therefore, its upper bound is $O(n)$.

$6^{\log_4 n}$ is $O(n)$, therefore, the upper bound for this problem is $O(n) + O(n) = MAX(O(n), O(n)) = O(n)$.

5.

Guess $T(n) = cn \log_2 n + 2$ Basecase: let $n = 1$, $T(1) = 2 \leq 3cn \log_2 n) + 2$, basecase holds. Assume the guessed upper bound holds for $\lfloor\frac{4n}{3}\rfloor$. When substitute it in the recurrence, we have

$$\begin{aligned}
T(n) &= 3T(\lfloor\frac{2n}{3}\rfloor+2)+\frac{2n}{3}\\
&\le 3(c\lfloor\frac{2n}{3}\rfloor\log_2\lfloor\frac{2n}{3}\rfloor+2)+n\\
&\le 3cn\log_2\lfloor\frac{2n}{3}\rfloor+6+n\\
&= 3cn(log_2n+\log_2\frac{2}{3})+6+n\\
&= 3cnlog_2n+3cn\log_2\frac{2}{3}+6+n\\
&= 3cnlog_2n+2+4+n-3cn\log_2\frac{3}{2}
\end{aligned}$$

To allow the last term $\le cn\log_2 n+2$, the assumed upper bound, we have choose c such that $4+n-3cn\log_2\frac{3}{2}\le 0$. let $c=\frac{2}{\log_2\frac{3}{2}}$,

$$4+n-3\cdot\frac{2}{\log_2\frac{3}{2}}\cdot\log_2\frac{3}{2}\cdot n = 4-5n\le 0$$

for all $n\ge 1$.
So, we have shown $T(n)\le cn\log_2 n+2$, for all $n\ge 1$.
So, the upper bound for this problem is $O(n\log_2 n)$.

6.

(a) REC-INSERTION SORT(A,n)

```
if $n > 1$
   then REC-INSERTION SORT(A,n-1)
   key = A[n]
   i = n - 1
   while $i > 0$ and A[i]>key
      A[i+1] = A[i]
      i = i - 1
   A[i+1] = key
```

(b) The recurrence for the time function is $T(n)=T(n-1)+(n-1)$ where $T(1)=0$.

$$\begin{aligned}
T(n) &= T(n-1)+(n-1)\\
&= T(n-2)+(n-2)+(n-1)\\
&= T(n-3)+(n-3)+(n-2)+(n-1)\\
&= ...\\
&= T(1)+1+2+3+...+(n-2)+(n-1)\\
&= \sum_{k=1}^{n-1}k
\end{aligned}$$

The upper bound for the arithmetic sequence is $O(n^2)$ which is a quadratic function in n.

(c) For the i^{th} elements, the comparisons are $\log_2 n$ but the shifts are remains n. So when we add them up, we have

$$\begin{aligned}\sum_{i=0}^{n}(i+\log_2 i) &= \frac{n(n+1)}{2}+\log_2(n!)\\ &= O(n^2+n\log_2 n)\\ &= O(n^2)\end{aligned}$$

In another word, the search (binary search) use $O(\log_2 n)$, and shift use $O(n)$. Total cost is $O(n)+O(\log_2 n) = O(n)$. To do this for $n-1$ elements, total cost is $n(n-1)$, which is $O(n^2)$.
For the best case, the array does not need to be moved, to the total cost is only the cost for search. To do this for $n-1$ elements, is $n \cdot \log_2 n$, which is $O(n \log_2 n)$.

7.

It can not work. Since the Heap is the binary tree, and only requirement is that the child node is less than parent. Therefore, if the root is 10, left child and right child are 7, 8. Then the algorithm will assume that the array is a sorted array but actually not.