$64/75 \approx 85\%$

Introduction to Algorithms PSET 1

1) $f_1 = \log(n^n) = n\log n$

$$\lim_{n\to\infty} \frac{f_1}{f_2} = \lim_{n\to\infty} \frac{n\log n}{(\log n)^n} = \lim \frac{n}{(\log n)^{n-1}} = 0.$$

$$f_5 = \log(\log(6006n)) = \log(\log 6006 + \log n)$$
$$\log(\log(6006n)) \sim \log(\log n)$$
$$= O(\log n)$$
$$= O(f_1, f_2, \ldots, f_4)$$

$$= \lim_{n\to\infty} (\log n)^{2-n} + (2-n)(\log n)^{1-n}$$
$$= \lim_{n\to\infty} (2-n)(\log n)^{1-n} = \lim_{n\to\infty} -n(\log n)^{1-n}$$
$$\Rightarrow f_1 = O(f_2)$$

$$\log(n^{6006}) = 6006\log n$$
$$= \Theta(\log n)$$
$$f_3 = O(f_1) \text{ and } O(f_2)$$

$$f_4 = (\log n)^{6006} = O(f_2)$$
$$f_3 = O(f_4)$$
$$\lim_{n\to\infty} \frac{f_4}{f_1} = \lim_{n\to\infty} \frac{(\log n)^{6006}}{n\log n} = \lim_{n\to\infty} \frac{(\log n)^{6005}}{n} = \lim_{n\to\infty} \frac{6005 \cdot (\log n)^{6004} \cdot \frac{1}{n}}{1}$$
$$= \lim_{n\to\infty} 6005 \cdot \frac{(\log n)^{6004}}{n}$$
$$= \lim_{n\to\infty} 6005! \, \frac{\log n}{n} = \lim_{n\to\infty} 6005! \, \frac{1}{n}$$
$$= 0.$$
$$\Rightarrow \lim_{n\to\infty} \frac{f_4}{f_1} = 0$$
$$\Rightarrow f_4 = O(f_1) \,//$$

$f_1 = O(f_2)$.

$f_3 = O(f_1)$ and $O(f_2)$ and $O(f_4)$.

$f_4 = O(f_1)$, $f_4 = O(f_2)$.

Ans: f_5, f_3, f_4, f_1, f_2 // ✓

Date No.

b)

Ans: $2^n,\ 6006^n,\ 6006^{2^n},\ 2^{6006^n},\ 6006^{n^2}$

$= f_1, \cancel{f_2, f_5, f_3} \cancel{f_4}$

c) $f_2 = \frac{n!}{(n-6)!\,6!} \sim \frac{n!}{(n-6)!} = n(n-1)(n-2)(n-3)(n-4)(n-5)$

$\sim n^6$

$f_3 = (6n)! \sim \sqrt{2\pi \cdot 6n}\left(\frac{6n}{e}\right)^{6n}$

$\sim \sqrt{n} \cdot 6^{6n} \cdot n^{6n} = 6^{6n} \cdot n^{6n-\frac{1}{2}}$

$f_4 = \binom{n}{n/6} = \frac{n!}{(n/6)!\,(n-\frac{n}{6})!} = \frac{n!}{(\frac{n}{6})!\,(\frac{5n}{6})!} \sim \frac{\sqrt{2\pi n}\left(\frac{n}{e}\right)^n}{\sqrt{2\pi \frac{n}{6}}\left(\frac{n}{6e}\right)^{\frac{n}{6}} \cdot \sqrt{2\pi \frac{5n}{6}}\left(\frac{5n}{6e}\right)^{\frac{5n}{6}}}$

$= \frac{\sqrt{n}\left(\frac{n}{e}\right)^n}{\sqrt{2\pi}\left[\sqrt{\frac{n}{6}}\left(\frac{n}{6e}\right)^{\frac{n}{6}} \cdot \sqrt{\frac{5n}{6}}\left(\frac{5n}{6e}\right)^{\frac{5n}{6}}\right]}$

$= \frac{\sqrt{n}\left(\frac{n}{e}\right)^n}{\sqrt{\frac{1}{3}\pi}\left[\sqrt{n}\left(\frac{n}{6e}\right)^{\frac{n}{6}} \cdot \sqrt{n}\left(\frac{5n}{6e}\right)^{\frac{5n}{6}}\right]}$

$\sim \frac{\sqrt{n}\left(\frac{n}{e}\right)^n}{\sqrt{n}\left(\frac{n}{6e}\right)^{n/6} \cdot \sqrt{n}\left(\frac{5n}{6e}\right)^{\frac{5n}{6}}}$

$\{f_2, f_5\} = O(f_1)$ $\quad = \frac{\left(\frac{1}{e}\right)^n \cdot \cancel{n^n}}{\sqrt{n} \cdot \cancel{n^{n/6}}\left(\frac{1}{6e}\right)^{n/6} \cdot \cancel{n^{5n/6}} \cdot \left(\frac{5}{6e}\right)^{\frac{5n}{6}}}$

$f_1 = O(f_3)$ $\quad = \frac{\left(\frac{1}{e}\right)^n}{\sqrt{n}\left(\frac{1}{6e}\right)^{n/6}\left(\frac{5}{6e}\right)^{\frac{5n}{6}}}$

$\{f_2, f_5\}, f_4, f_1, f_3$ $\quad = \frac{\left(\frac{1}{e}\right)^n}{\sqrt{n}\left(\frac{1}{6}\right)^{n/6}\left(\frac{1}{e}\right)^{n/6}\left(\frac{5}{6}\right)^{\frac{5n}{6}}\left(\frac{1}{e}\right)^{\frac{5n}{6}}}$

$= \frac{\left(\frac{1}{e}\right)^n}{\sqrt{n}\left(\frac{1}{6}\right)^{n/6}\left(\frac{5}{6}\right)^{5n/6}\left(\frac{1}{e}\right)^n}$

$= \frac{1}{\sqrt{n}\left(\frac{1}{6}\right)^{n/6}\left(\frac{5}{6}\right)^{5n/6}} = \frac{1}{\sqrt{n}\left(\frac{1}{6}\right)^{n/6}\left(\frac{5^5}{6^5}\right)^{n/6}}$

$= \frac{1}{\sqrt{n}\left(\frac{5^5}{6^6}\right)^{n/6}}$

$\sim \frac{1}{\sqrt{n}\left(\frac{1}{6}\right)^{n/6}} = \frac{1}{\sqrt{n}\,6^{-n/6}}$

$= \frac{6^{n/6}}{\sqrt{n}}$

$= \frac{e^{\frac{n}{6}\ln 6}}{\sqrt{n}}$

Date No.

d) $f_1 = n^{n+4} + n! = n^{n+4} + \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$\tilde{\ } \; n^n\left(n^4 + \sqrt{n}\left(\frac{1}{e}\right)^n\right)$

$= n^{n+\frac{1}{2}}\left(n^{3.5} + \left(\frac{1}{e}\right)^n\right)$

$\sim n^{n+\frac{1}{2}}\left(n^{3.5}\right) = n^{n+4} = 2^{(n+4)\log n}$

$f_2 = n^{7 \cdot n^{\frac{1}{2}}} = 2^{7n^{\frac{1}{2}}\log n}$

$\lim_{n\to\infty} \frac{f_1}{f_2} = \frac{n^{n+4}}{n^{7n^{\frac{1}{2}}}} = n^{n+4-7\sqrt{n}} = \infty$

$\Rightarrow f_2 = O(f_1)$

$f_3 = 2^{6n\log n}$ $\lim_{n\to\infty} \frac{f_1}{f_3} = \lim_{n\to\infty} 2^{(n+4-6n)\log n} = \lim_{n\to\infty} 2^{(4-5n)\log n}$

$= \lim_{n\to\infty} 2^{4\log n - 5n\log n}$

$= \lim_{n\to\infty} \frac{2^{4\log n}}{2^{5n\log n}}$

$= 0$

$\Rightarrow f_1 = O(f_3)$

$\Rightarrow f_2 = O(f_3)$

$f_4 = 7^{n^2} \Rightarrow 2^{2n^2} < f_4 < 2^{3n^2}$

$f_1, f_2, f_3 = O(f_4)$.

$f_5 = n^{12+\frac{1}{n}} \sim n^{12} = O(f_1, f_2, f_3, f_4)$.

f_5, f_2, f_1, f_3, f_4 //

19

1-2)

We use recursion.
The algorithm is as follows:

If $k < 2$, then return as there are no objects to reverse.
else,
delete the i item and store it in a variable.
data = D.delete_at(i).
insert data into the $i+k-1$ index.
D. insert_at ($i+k-1$, data).
recurse again, this time starting at i for the next $k-1$ items.
reverse (D, i, $k-1$).

By induction, we induct on the number of items to reverse, k.

Base case:
When $k=0$, the algorithm returns and nothing changed. as no items to reverse. ✓.

Inductive step:
Assume that the algorithm can successfully reverse n items, where $0 \le n \le k$, then for $n+1$ items,
Algorithm must reverse items from index i to index $i+n$.
Let x be the last item to reverse. x is at $i+n$.

Algorithm deletes the i^{th} item. Let the deleted item be d.
Now, x is at index $i+n-1$, as one item before it was removed.

Place d in the $i+n^{th}$ index, infront of the $i+n-1^{th}$ item.
d is now infront of the last item to be reversed. d is in the correct location.
Now, we are left with reversing the items from i to $i+n-1$.

There are a total of $i-i+n+1-1=n$ items to reverse.
By the I.H, these items will be successfully reversed.

$\therefore$ Algorithm works.

□.

For each item, we call one delete_at and one insert_at.
Moving one item has running time of two $O(\log n)$ operations.
$\Rightarrow$ one item takes $O(\log n)$ time.
$\therefore$ Moving k items take $O(k \log n)$ time.

b) def move (D, i, k, j):
    if k == 0, then return.

    l = i+k   l is the index of the last item to move
    d = D.delete_at(l)
    D.insert_at(j+1, d)   last item is now infront of item at j.
    move(D, i, k-1, j)   recurse on the remaining k-1 items.

Pf: We induct on k, the number of items.

Base case: k = 0. Algorithm does nothing as no items to move. ✓.

Inductive step:

Suppose algorithm works for n items, $0 \le n \le k$,
then for n+1 items,

Let d be the last item at index i+n+1
we move d to be at index j+1, infront of j, behind the item previously at j+1.
Now there are items from i to i+n that we must move.
There are n items to move.
By the I.H, these items can be moved infront of j using the algorithm, with the first item to be moved at j+1, and last at j+1+n.

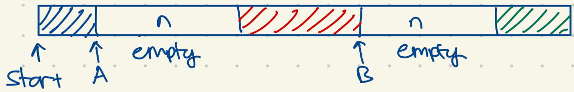
Since d was at j+1, d will now be at j+1+n as all items were inserted infront of j and behind j+1.
Now, all items are infront of j.
∴ Algorithm works.

□.

Algorithm also uses O(k log n) time as it uses same amt of fn calls as reverse.

16

1-3) Using an array of length $3n$, we can store items behind bookmark A at the start of the array, leave a n length gap, store items from A to B after the gap, leave another n length gap, and store items after B in the remaining locations.



We define a partition of the pages to be a sequence in the array containing page data. Each partition has two pointers. One points to the index of the beginning of the partition, and the other points to the end of the partition.
For the kth partition, let k_1 be the start index of the partion. k_2 be the end.

We define a bookmark to be a variable storing the index of the page it is infront of
e.g bookmark at i is between page at index i and index $i+1$.
Let the value of bookmark A be A, and similarly for B.

Upon building, initialize an array of length $3n$ and store the pages from index 0 to $n-1$. This is the first partition. Let the start and ends be a_1, a_2.

For the first place_mark call,
When placing the first mark at i, move items from a_1+i+1 to a_2 to fill indexes from $2n+i+1$ to $3n-1$. There are now two partitions. Let the second partition be c_1, c_2.

For the second place_mark call,

<u>Case 1</u>
When placing the second mark at page k, if $k < a_2$, then the mark must partition the 1st partition. Move all items from index a_1+k+1 to a_2 to fill the indexes $a_2'+n+1$ to $a_2'+n+k$. where a_2' is the new end of partition one.
Let this partition be b_1, b_2

<u>Case 2</u>
For the second place_mark call,
if $k > a_2$, then mark must partition the 2nd partition. Let $\ell = k - a_2 + 1$ be the number of items to move.
Move items from index c_1 to $c_1 + \ell - 1$ to indexes a_2+n+1 and $a_2+n+\ell-1$

Let this partition be b_1, b_2.

Now the pages are partitioned into 3, with a gap of n between each partition.

This all is in $O(n)$.

To support read_page,

Let P_1 be partition one. $|P_1| = a_2 - a_1 + 1$.
P_2 be partition two. $|P_2| = b_2 - b_1 + 1$.
P_3 be partition three. $|P_3| = c_2 - c_1 + 1$.

For reading i, if $i < |P_1|$ then i is in partition 1.
return item at $a_1 + i$.

If $|P_1| \leq i < |P_1| + |P_2|$, then i is in P_2.
return item at $b_1 + (i - |P_1|)$.

If $|P_1| + |P_2| \leq i < |P_1| + |P_2| + |P_3|$, i is in P_3.
return item at $c_1 + (i - |P_1| - |P_2|)$

This is in $O(1)$

To support shift_mark,

If A or B are at the ends of the partitions, and will be moving into an index with no data, then move the book mark to the adjacent start/end of the next partition.
e.g suppose $A = b_1$. shift_mark$(A, -1)$.
Since $b_1 - 1$ is empty,
$A = a_2$ as that is the previous page from b_1.

Similarly if $B = b_2$, shift-mark$(B, 1)$, then
$B = c_1$ as $b_2 + 1$ is empty.

If A, B are not at ends, simply increment or decrement the bookmark.

Operation in $O(1)$ time.

So complicated compared to the answer.

To support move_page,
First, we must 'fix' the arrangement of pages, since the bookmarks might be in other partitions, and we want the partitions to reflect the bookmark positions in the page.

We first fix B. If $B \neq b_2$,
if B in P_3, move pages from c_1 to B to the end of P_2. Set B to b_2.
If B in P_2, move pages from $B+1$ to b_2 to the start of P_3.

Now for A bookmark. If $A \neq a_2$,
then if A is in P_2, move pages at b_1 to A to the end of P_1, and reset A to a_2.
if A is in P_1, move pages from $A+1$ to a_2 to the start of P_2.
if A in P_3, move P_2 to the front of P_1, and move $A+1$ to B to index $a_2 + n$. Set A to a_2.

This is $O(n)$ in the worst case. Since shift_mark is $O(1)$, n operations of shift mark, we move n items using move_page() hence move page is $O(1)$ amortized.

I don't think you can use shift_mark() to charge move_page()

12

4) def insert_first(self, x):
head = self.head
x.next = head. set next pointer of first node to the old head
head.prev = x. Have the old head point back to x.

self.head = x. Set the head of the list to the new head.
return O(1) since each function call takes constant time regardless of the length of the list.

```
def insert_last(self, x)
    tail = self.tail.
    x.prev = tail.
    tail.next = x.

    self.tail = x.
    return
```

```
def delete_first(self, x):
    head = self.head
    new_head = head.next.
    new_head.prev = null
    delete head.
    self.head = new_head.
    return.
```

```
def delete_last(self, x):
    tail = self.tail
    new_tail = tail.prev
    new_tail.next = null
    delete tail
    self.tail = new_tail
    return.
```

8

b) Let y_1 be the node before x_1
y_2 be the node after x_2. case for if x_1, x_2 are heads, tails?

Store pointers to y_1, y_2, x_1, x_2.
Detach x_1 from y_1, and x_2 from y_2, by clearing x_1.prev, y_1.next, x_2.next, y_2.prev.
Connect y_1 to y_2. by setting y_1.next = y_2.
y_2.prev = y_1.

Now x_1, x_2 are the head and tail of a linked list in memory. We must make a container to reference x_1, and x_2.

Let M be a new empty linked list. L.head = x_1. L.tail = x_2
return M, for a new doubly linked list from $x_1, \dots, x_2$.

c) Let y be the node after x, x.next. Same, if no x.next?

Extract head and tail of L_2 and store it in z_1, z_2 respectively.
Clear the head and tail of L_2 to make L_2 empty.

4

Let z_2.next = y.
z_1.prev = x.
x.next = z_1

Now L_1 is has the linked list
head ⟺ … ⟺ x ⟺ z_1 ⟺ … ⟺ z_2 ⟺ y ⟺ … ⟺ tail.

5

And L_2 has no list since head and tail is empty.
Regardless of the amt of nodes in L_1, L_2, algorithm only operates a fixed no. of operations on x_1, y, z_1, z_2 ⟹ O(1) time.

17