HW5

1. Recurrence Relations

Since it's trivial to draw the work-tree method form or the tree for this sort of problem, I'll just do it once for the first problem and then give the answer directly.

i. The recurrence relation is $T(n) = 5T\left(\frac{n}{2}\right) + O(n)$.

Node size	Number of nodes	Work on node	Total work
S=n	1	c*s	c*n
S=n/2	5	c*s	5*c*n/2
S=n/4	25	c*s	25*c*n/4
…	…	…	…

Adding up the total work at each level of recursion gives us:

$$S_{total} = cn + 5c\frac{n}{2} + 5^2 c\frac{n}{2^2} + cn\frac{5^3}{2^3} + \ldots$$

Which can be rewritten as follow, where $n = 2^k \Leftrightarrow k = \log_2 n$:

$$cn\left(1 + \frac{5}{2} + \frac{5^2}{2^2} + \frac{5^3}{2^3} + \frac{5^4}{2^4} + \cdots + \frac{5^k}{2^k}\right) = \frac{2cn}{3}\left(\frac{5^k}{2^k} - 1\right)$$

$$= \frac{2cn}{3}\left(\frac{5^{\log_2 n}}{n}\right) = \frac{2cn}{3}\left(\frac{n^{\log_2 5}}{n}\right) = \frac{2c}{3}n^{\log_2 5} \in O\left(n^{2.32}\right)$$

ii. The recurrence relation is $T(n) = 2T(n-1) + O(1)$

The total work is $c\sum_{k=0}^{n} 2^k$

Which has an order of $S_{total} \in \left(2^n\right)$

iii. The recurrence relation is $T(n) = 9T\left(\frac{n}{3}\right) + O\left(n^2\right)$

The total work is

$$cn^2\left(1 + \frac{9}{3^2} + \frac{9^2}{3^4} + \frac{9^3}{3^6} + \frac{9^4}{3^8} + \cdots + \frac{9^k}{3^{2k}}\right) = cn^2 \log_3 n \in O\left(n^2 \log n\right)$$

Therefore, the boss should implement Algorithm C, since logarithms are always smaller than powers. (logn is upper bounded by $n^{0.32}$)

2. The Index Problem

```
def binary_search(array):
    high = len(array) - 1
    low = 0
```

```
while low <= high:
    mid = (low + high) // 2
    if array[mid] == mid:
        return mid
    elif array[mid] < mid:
        low = mid + 1
    else:
        high = mid - 1
return -1
```

This is basically a binary search problem. The program is written in Python style. It returns *i* if it exists and returns *-1* if it doesn't.

For each loop cycle, we divide the middle pointer's index by half, and it's just like a tree structure: each time we choose either larger or smaller branch. And it's clear that such a tree has a depth of (logn)

3. Recursive Calls

For this problem, I would use the work-tree form again to show why it's not an effective optimization.

i. The recurrence relation is $T(n) = 8T\left(\frac{n}{2}\right) + cn^4$

Node size	Number of nodes	Work on node	Total work
S=n	1	$c*s^4$	$c*n^4$
S=n/2	8	$c*s^4$	$8*c*(n/2)^4$
S=n/4	64	$c*s^4$	$64*c*(n/4)^4$
…	…	…	…

Which can be rewritten as follow, where $n = 2^k \Leftrightarrow k = \log_2 n$:

$$cn^4\left(1+\frac{8^1}{16^1}+\frac{8^2}{16^2}+\frac{8^3}{16^3}+\frac{8^4}{16^4}+\cdots+\frac{8^k}{16^k}\right) = 2cn^4\left(1-\frac{1}{n}\right)$$

ii. The recurrence relation is $T(n) = 7T\left(\frac{n}{2}\right) + dn^4$

Node size	Number of nodes	Work on node	Total work
S=n	1	$d*s^4$	$d*n^4$
S=n/2	7	$d*s^4$	$7*d*(n/2)^4$
S=n/4	49	$d*s^4$	$49*d*(n/4)^4$
…	…	…	…

Which can be rewritten as follow, where $n = 2^k \Leftrightarrow k = \log_2 n$:

$$cn^4\left(1+\frac{7^1}{16^1}+\frac{7^2}{16^2}+\frac{7^3}{16^3}+\frac{7^4}{16^4}+\cdots+\frac{7^k}{16^k}\right)=\frac{16}{9}cn^4\left(1-n^{-1.19}\right)$$

As we compare the overall runtime of these two algorithms, we found out that although it was optimized somehow, but both of these two have a same order at $O(n^4)$. None of these change the big-O, since the n^4 dominates its complexity.

4. Stooge Sort

i. For problem a, let's first discuss the base cases.
If array only has one element, then it immediately returns. If it contains two elements, the algorithm checks its order and swap if needed, then returns.
After we sort the first 2/3 of the array, we can always say that the first 1/3 of the array is smaller than the middle 1/3.
Then we sort the last 2/3 of the array, which makes the last 1/3 is larger than both the middle 1/3 and the first 1/3 of the array.
Finally we sort again the first 2/3 of the array, to make sure that the first 1/3 is smaller than middle 1/3. Therefore, we got the whole array sorted.

ii. The recurrence relation is $T(n)=3T\left(\frac{2n}{3}\right)+O(1)$

iii. We would assume that $n=\left(\frac{2}{3}\right)^k \Leftrightarrow k=\log_{\frac{3}{2}} n$

Node size	Number of nodes	Work on node	Total work
S=n	1	c*1	C
S=2n/3	3	c*1	3*c
S=4n/9	9	c*1	9*c
…	…	…	…

That gives us a

$$c\left(1+3+3^2+\cdots+3^k\right)=2c\left(3^k-1\right)=2c\left(3^{\log_{\frac{3}{2}} n}\right)=2c\left(n^{\log_{\frac{3}{2}} 3}\right)\in O\left(n^{2.709}\right)$$

Actually, it's even worse than bubble sort!

iv. Comparing with the other sorting algorithms, this is even slower than most of them. Selection, Bubble and Insertion all have a worst $O(n^2)$, while other better ones have O(nlogn). The only exception comes with Bogo sort which has factorial complexity.