Week 5 tute solutions:

Q1) Quick select average case

Theorem: Randomised Pivot Selection

Using randomised pivot selection, the Quickselect algorithm has expected run time $O(N)$.

Proof

Assume without loss of generality that the array contains no duplicate elements and denote by T_N the time taken by Quickselect to select the k'th order statistic of an array of size N. If the pivot selected is the i'th order statistic of the array, then the time taken by Quickselect will be

$$T_N = N + 1 + \begin{cases} T(N-i) & \text{if } i < k, \\ 1 & \text{if } i = k, \\ T(i-1) & \text{if } i > k \end{cases}.$$

Averaging out over all possible pivot selections, we obtain an expected run time of

$$T_N = N + 1 + \frac{1}{N}\left(\sum_{i=1}^{k-1} T(N-i) + \sum_{i=k+1}^{N} T(i-1) + 1\right).$$

Multiplying both sides by N, we obtain

$$NT_N = N^2 + N + \sum_{i=1}^{k-1} T(N-i) + \sum_{i=k+1}^{N} T(i-1) + \frac{1}{N}.$$

Substituting $N = N-1$, we find the equation

$$(N-1)T_{N-1} = (N-1)^2 + (N-1) + \sum_{i=2}^{k} T(N-i) + \sum_{i=k+1}^{N+1} T(i-1) + \frac{1}{N-1}.$$

Subtracting this from the earlier equation yields

$$NT_N - (N-1)T_{N-1} = N^2 + N - (N-1)^2 - (N-1) - T(N-1) + T(N-1) + \frac{1}{N} - \frac{1}{N-1}.$$

Cleaning up, we obtain

$$NT_N = (N-1)T_{N-1} + 2N + \frac{1}{N(N-1)}$$

Divide through by N and we find

$$T_N = \frac{N-1}{N}T_{N-1} + 2 + \frac{1}{N^2(N-1)}.$$

Finally, we use the facts that $(N-1)/N < 1$ and $1/(N^2(N-1)) < 1$ to state the bound

$$T_N < T_{N-1} + 3.$$

Recalling Lecture 2, the solution to this recurrence is the linear equation, hence we have established that the expected run time of Quickselect with randomised pivoting is linear.

$$T_N < 3N = O(N).$$

Q2.
What is the computational complexity of this algorithm? Attempt to prove it formally!

T_1 = 1
T_2 = 1
T_n = T_n-1 + T n-2
T_n < 2 * T n-1 , given that T N-1 > T N-2
T_n < 2^2 * T n-2
T_n < 2^3 * T n-3
...
T_n < 2^N * 1
Hence the complexity of this algorithm is bounded by O(2^n)

Can you write a more efficient version that is NOT iterative, but instead single-recursive (rather than double-recursive as in the version above)?

```
def fib(n):
     return _fib(n, 0, 1)

def _fib(n, a, b):
    if n ==0:
        return b
    return _fib(n-1, b, a+b)
```

What is the time complexity of such a single-recursive implementation?

O(n)

Q3) How many times is body() executed for the following values?

Lo1=1, Hi1=10, Lo2=i, Hi2=10 --- 55
Lo1=0, Hi1= 9, Lo2=0, Hi2=9 --- 100
Lo1=1, Hi1=n, Lo2=i-2, Hi2=i+2 --- 5n
Lo1=0, Hi1=n, Lo2=i, Hi2=2*i --- See below.
T0 = 1
T1 = T0 + 2 = 3
T2 = T1 + 3 = 6
T3 = T2 + 4 = 10
T3 = T1 + 3 + 4
T3 = T0 + 2 + 3 + 4
T3 = 1 + 2 + 3 + 4
Just a sum of natural numbers to N + 1, so
T_N = ((N + 1) (N + 1 + 1)) / 2
T_N = (N^2 + 3N + 2) / 2

Q4)

Given an integer n, body() will be run n times where n > 0, and base() will always run once. function r(n), given an integer of n > 0 will make a single call to body() before recursively calling itself with the value of n-1. As our n value shrinks by 1 every time our function is recursed, n will be greater than zero on n occasions, calling body() on n occasions. No matter what value n our function r is given (even if a negative number), the "else" condition will yield true on exactly one occasion.

Thus,

r(10), body() = 10, base() = 1

r(5), body() = 5, base() = 1

r(1), body() = 1, base() = 1

Q5)

If M_i is the minimum of coins to form the value = i, using the denomination set $\{D_1, \ldots, D_N\}$.

Initialize $M_0 = 0$

DP recurrence:

$M_i = \min_{1 \le j \le N} (M_{i-D_j} + 1)$ for all $1 \le i \le V$