# CPSC413 Assignment 2

Yifan Chen

30073072

## 1 Q1. Deviations in running times of various algorithms

1. $n^2$
   a. $(2n)^2 = 4n^2$, the runtime becomes 4 times slower.
   b. adding 1 to the input will have no significant effect on runtime.

2. $n^3$
   a. $(2n)^3 = 8n^3$, the runtime becomes 8 times slower.
   b. adding 1 to the input will have no significant effect on runtime.

3. $100n^2$
   a. $100(2n)^2 = 400n^2$, the runtime becomes 4 times slower.
   b. adding 1 to the input will have no significant effect on runtime.

4. $nlogn$
   a. $(2n)log(2n)$, the runtime will be slightly more then 2 times slower.
   b. adding 1 to the input will have no significant effect on runtime.

5. $2^n$
   a. $2^{2n}$, the runtime will double exponentially.
   b. $2^{n+1} = 2 * 2^n$., the runtime becomes 2 times slower.

## 2 Q2. Arrange the following list of functions in ascending order of growth rate

$$f_2 \rightarrow f_3 \rightarrow f_6 \rightarrow f_1 \rightarrow f_4 \rightarrow f_5$$

**Reasoning**: Asymptotically, $\sqrt{n} < n < n^2 log(n) < n^{2.5}$. $f_4$ and $f_5$ both growth exponentially which is larger then the polynomial growth rate of the other 4 functions, and $10 < 100$.

## 3 Q3 Prove the following statements

First recall that the definition of big O, big $\Omega$ and big $\Theta$
$T = O(f) \iff \exists c > 0, n_0 \geq 0 \forall n \geq n_0 : T(n) \leq c * f(n)$
$T = \Omega(f) \iff \exists \epsilon > 0, n_0 \geq 0 \forall n \geq n_0 : T(n) \geq \epsilon * f(n)$
$T = \Theta(f)$ if and only if $T = O(f)$ and $T = \Omega(f)$
$T = \Theta(f)$ if limit as x approaches $\infty$ of $T(n)/f(n) = c, c > 0$.

1. $2n^2 + \sqrt{n} = \Omega(n)$
   **Proof.** Suppose a constant $\epsilon = 1$, there then exists $n_0$, such that $2n^2 + \sqrt{n} \geq \epsilon * n$ for all $n \geq n_0, n_0 \geq 0$
   $2n^2 + \sqrt{n} \geq n$, which holds true for all $n \geq 0$, thus we can say
   for $\epsilon = 1, 2n^2 \geq \epsilon * n$, for all $n \geq 0$.

2. $5n^3 + 3.5n^2 - 7n + 19 = O(n^3)$
   **Proof.**
   $5n^3 + 3.5n^2 - 7n + 19 \leq 5n^3 + 3.5n^3 - 7n^3 + 19n^3$
   $= (5 + 3.5 - 7 + 19)n^3 = O(n^3)$

3. $n^4 = O(2^n)$
   **Proof.** Suppose a constant $c = 1$, there then exists $n_0$, such that $n^4 \leq 2^n * c$ for all $n \geq n_0, n_0 \geq 0$.
   $n^4 \leq 2^n$
   by inspection, we see $n \geq 16$.
   So for c = 1, and $n \geq 16$, $n^4 \leq c * 2^n$

4. $20n^2 + nlogn = \Theta(n^2)$
   $\lim_{x\to\infty}((20n^2+nlogn)/n^2) = \lim_{x\to\infty}((20+nlogn/n^2)/1) = 20 + \lim_{x\to\infty} logn/n$
   , using L'hopitals, assuming ln = 20.

   thus since the limit converges to a real constant, $20n^2 + nlogn = \Theta(n^2)$

# 4 Q4. Rewrite Gale-Shapely to explicitly use the described data structures

a. The algorithm would require an array $W_x$ that maintains the set of women to which men have not yet proposed to. It would also require 2 more arguments, 2 arrays,$WP_y[1...n]$ and $MP_x[1...n]$, which contain the preference lists of each man and woman, where $WP_y[i]$ stores the index of the ith man in y's ranking, and $MP_x[i]$ stores the index of the ith woman in x's ranking. The set of freemen will be put into a linked list of integers. An integer array next[] will be used to point to the next woman that m will propose to next, in order of the man's preference. Integer Array Current[w], is set to 0 for all women, and when a woman becomes engaged, it is set to the man that proposed. Finally a 2d array ranking will be created off womans preference list.
b. To denote that all men are free initially, add the set of all men to the linked list. To denote that all woman are free, the array Current[w] should all be initalized to 0. The cost of each operation is O(n).
c.run until the stack of men is empty, if there is a freeman, he will be in the stack. The cost of finding the man would be O(1).
d.We will choose the head of the list, and pop him from the stack, the cost of this will be O(1)
e. Get the highest ranked woman on m's list, and set it to w. increment next so it points to the next highest woman in m's preference list. The cost of this implementation will be O(1).
f. To check if a woman is free, we maintain yet another array $Current[]$, which stores x, man she is engaged to. If not yet engaged, $Current[0]$. The cost of checking this condition will be O(1)
g. They will be kept track of in the array current, Current[w] will be set to m, the cost of making the pair will be O(1).
h. w will be engaged to w's current partner m' in the array Current[].To determine who w prefers, we will use array $rank_y[1...n]$ which identifies the rank of the man m in w's preference list. The array will compare whether w prefers m or m', and will choose the higher ranking one. The 2d array will also allow us to operate in constant time, hence, the cost of the operation is O(1).
i. m will be pushed back into the linked list, the cost of this is O(1).
j. Same as 6, the cost will be the same as well.
k. Current[w] would need to be replaced with m, and m' would need to be added back into the linked list. cost of this operation is O(1) + O(1)
l. Return the array current. cost will be O(1).
m. Costs of lines 2-12 is some constant c, loop is executed at worst-case $n^2 - n + 1$ times. Line 1 costs the initialization of several arrays of size n, denoted by integer d, aswell as 2d ranking array of size $n^2$. Line 16 cost 1, so $c * (n^2 - n + 1) + n^2 + dn + 1$.
n. runtime = $\Theta(n^2)$, $\lim_{x\to\infty}((c * (n^2 - n + 1) + n^2 + dn + 1)/n^2) = c + 1$, and since the limit exists as a positive constant, by the definition the bound is correct.

# 5 Give a worst case analysis of Binary Search

a. Assuming A[j] = A[j+1] and A[k] = X[i] takes a constant number of steps, we get $\sum_{i=1}^{n-1}(log(n)+c2i) = (n-1)*log(n)+(n-1)*2c(n+1)/2 = nlogn+n$.
b. let c = 2, $nlogn + n \leq c * nlogn$ for all n, $n \geq 0$, so $O(nlogn)$
c. let c = 1, $nlogn + n \geq c * nlogn$ for all n, $n \geq 0$, so $O(nlogn)$
d. Since $O(nlogn)$ and $\Omega(nlogn)$, $\Theta(nlogn)$