Xi Liu, xl3504, Homework 1

for this entire assignment, I use “IH” as acronym for “inductive hypothesis”

Problem 1
let $p(n)$ be the proposition that

$$1 \times 1! + 2 \times 2! + ... + n \times n! = (n+1)! - 1 \qquad \forall n \in \mathbb{N}$$

1.
base step:
$p(1)$ is true, when $n = 1,\ 1 \times 1! = 1 = (1+1)! - 1 = 2 - 1 = 1$

2.
inductive step:
assume $p(k)$ is true for some positive integer k, or equivalently, assume

$$1 \times 1! + 2 \times 2! + ... + k \times k! = (k+1)! - 1$$

is true

$$\begin{aligned}
&1 \times 1! + 2 \times 2! + ... + k \times k! + (k+1) \times (k+1)! \\
&\stackrel{\text{IH}}{=} ((k+1)! - 1) + (k+1) \times (k+1)! \\
&= (k+1)! - 1 + (k+1) \times (k+1)! \\
&= (k+2)(k+1)! - 1 \\
&= (k+2)! - 1 \\
&= ((k+1)+1)! - 1
\end{aligned}$$

so $p(k+1)$ is true
by mathematical induction, $p(n)$ is true $\forall n \in \mathbb{N}^+$

Problem 2
let $p(n)$ be the proposition that

$$a_n = 2^n + 1$$

1.
base step:
$p(1)$ is true, when $n = 1, a_1 = 2^1 + 1 = 3$
$p(2)$ is true, when $n = 2, a_2 = 2^2 + 1 = 5$

2.
inductive step:
assume $p(i)$ is true for some positive integers $i,\ k$ such that $1 \leq i \leq k$, in particular for $i = k$ and $i = k - 1$:

$$a_k = 2^k + 1$$
$$a_{k-1} = 2^{k-1} + 1$$

is true

$$\begin{aligned} a_{k+1} &= 3a_k - 2a_{k-1} \\ &= 3(2^k + 1) - 2(2^{k-1} + 1) \\ &= 3 \cdot 2^k + 3 - 2^k - 2 \\ &= (3 - 1) \cdot 2^k + 3 - 2 \\ &= 2 \cdot 2^k + 1 \\ &= 2^{k+1} + 1 \end{aligned}$$

so $p(k + 1)$ is true
by mathematical induction, $p(n)$ is true $\forall n \in \mathbb{N}^+$

Problem 3

the order is:

(e) $2^{\log_2 n},\ (b)\ n^{1.5}\log_2 n,\ (c)\ n^2-1,\ (a)\ 2^n,\ (f)\ 3^n,\ (d)\ n!$

Problem 4

a.

false

let $f(n) = n$, $g(n) = n^2$

$f(n) = O(g(n))$ or equivalently $n = O(n^2)$ because there exist positive constants c and n_0 such that $0 \leq n \leq cn^2$ for all $n \geq n_0$

$$n \leq cn^2$$

dividing by n yields

$$1 \leq cn$$

we can make the inequality hold for any value of $n = n_0 \geq 1$ by choosing any constant $c \geq 1$

claim: but $g(n) \neq O(f(n))$ because $n^2 \neq O(n)$

proof: for contradiction, assume $n^2 = O(n)$, then there exist positive constants c and n_0 such that $0 \leq n^2 \leq cn$ for all $n \geq n_0$

$$n^2 \leq cn$$

dividing by n yields

$$n \leq c$$

which cannot remain true for arbitrary large n, since c is a constant

b.

true

$f = \Omega(h)$ means $f(n) \geq c_1 h(n)$ for all $n \geq n_1$, when c_1 and n_1 are some positive constants

$g = \Omega(h)$ means $g(n) \geq c_2 h(n)$ for all $n \geq n_2$, when c_1 and n_1 are some positive constants

$$(f + g)(n) = f(n) + g(n) \geq c_1 h(n) + c_2 h(n) = (c_1 + c_2) h(n)$$

when $n \geq n_1$ and $n \geq n_2$, let $c_3 = c_1 + c_2$

since

$$(f + g)(n) \geq c_3 h(n)$$

$$(f+g)(n)=\Omega(h(n))$$

c.
true
$f=O(g)$ means $f(n)\leq c_1 g(n)$ for all $n\geq n_1$, when c_1 and n_1 are some positive constants
$g=O(h)$ means $g(n)\leq c_2 h(n)$ for all $n\geq n_2$, when c_2 and n_2 are some positive constants
when $n\geq n_1$ and $n\geq n_2$, multiply $g(n)\leq c_2 h(n)$ by c_1 yields

$$c_1 g(n)\leq c_1 c_2 h(n)$$

so

$$f(n)\leq c_1 g(n)\leq c_1 c_2 h(n)$$

let $c_3=c_1c_2$, then

$$f(n)\leq c_3 h(n)$$

so

$$f(n)=O(h(n))$$

d.
false
let $g(n)=n$, $f(n)=n^2$, $h(n)=n^3$
$f=\Omega(g)$ or equivalently $n^2=\Omega(n)$, since $n^2\geq c_1 n$ for all $n\geq n_1$, when $c_1\geq 1$ and $n_1\geq 1$
$h=\Omega(g)$ or equivalently $n^3=\Omega(n)$, since $n^3\geq c_2 n$ for all $n\geq n_2$, when $c_2\geq 1$ and $n_2\geq 1$
but $f\neq\Omega(h)$ or equivalently $n^2\neq\Omega(n^3)$, since $n^2\ngeq c_3 n^3$ or equivalently $n^2<c_3n^3$ for all $n\geq n_3$, when $c_3\geq 1$ and $n_3>c_3$

Problem 5

(a)

```
SELECTION_SORT(A)
    for i = A.length downto 2
        max_i = i;
        for j = i - 1 downto 1
            if(A[j] > A[max_i])
                max_i = j
        max = A[max_i]
        for k = max_i + 1 to i
            A[k - 1] = A[k]
        A[i] = max
```

(b)

best case: $\Theta(n^2)$

worst case: $\Theta(n^2)$

since there are 2 levels of nested for loops and the lines within the innermost level take constant time

(c)

the variant of SELECTION_SORT that finds the 2 largest elements among the remaining elements and place them at the end is not more time-efficient to the original SELECTION_SORT

for this variant of SELECTION_SORT: from the code below, although the number of outer for loop is approximately half of the number of original for loop (since i is subtracted by 2 in each iteration of the for loop), but inner for loop created by FIND_2MAX have more assignment operations and comparisons

because for the variant of SELECTION_SORT, there are also 2 levels of nested for loops and the lines within the innermost level take constant time, so the variant of SELECTION_SORT that finds the 2 largest elements have complexity:

best case: $\Theta(n^2)$

worst case: $\Theta(n^2)$

```
SWAP(A, i, j):
```

```
    t = A[i]
    A[i] = A[j]
    A[j] = t

FIND_2MAX(A)
    fst = 0
    sec = -1
    for i = 1 to A.length
        if A[i] > A[fst]
            sec = fst
            fst = i
        else if A[i] < A[fst]
            if sec == - 1 or A[i] > a[sec]
                sec = i
        else if A[i] == A[fst] and i != fst
            sec = i
    return fst, sec

SELECTION_SORT(A):
    for i = A.length downto 1 by -2
        fst, sec = FIND_2MAX(A, i + 1)
        SWAP(A, i, fst)
        if sec != i
            SWAP(A, i - 1, sec)
```

Problem 6

COMPARE-1 returns TRUE when every sum of paired elements $A[i] + C[i]$ is less than $B[j]$ for all $1 \leq i \leq n,\ 1 \leq j \leq n$

for example, if the function is called with the below input, then it will return TRUE

```
A = {0, 1, 2}
C = {0, 1, 2}
B = {5, 6, 7}
n = 3
```

since

```
A[1] + C[1] = 0 + 0 = 0 < B[1] = 5
A[1] + C[1] = 0 + 0 = 0 < B[2] = 6
A[1] + C[1] = 0 + 0 = 0 < B[3] = 7
A[2] + C[2] = 1 + 1 = 2 < B[1] = 5
A[2] + C[2] = 1 + 1 = 2 < B[2] = 6
A[2] + C[2] = 1 + 1 = 2 < B[3] = 7
A[3] + C[3] = 2 + 2 = 4 < B[1] = 5
A[3] + C[3] = 2 + 2 = 4 < B[2] = 6
A[3] + C[3] = 2 + 2 = 4 < B[3] = 7
```

COMPARE-2 returns TRUE when the maximum value of $A[i] + C[i]$ is less then $B[j]$ for all $1 \leq i \leq n,\ 1 \leq j \leq n$

so COMPARE-2 will return the same boolean value as COMPARE-1 if given the same input arrays
using the same input as above, if the function is called with the below input, then it will return TRUE

```
A = {0, 1, 2}
C = {0, 1, 2}
B = {5, 6, 7}
n = 3
```

since

aux = A[3] + C[3] = 2 + 2 = 4 < B[1] = 5
aux = A[3] + C[3] = 2 + 2 = 4 < B[2] = 6
aux = A[3] + C[3] = 2 + 2 = 4 < B[3] = 7

(b)
COMPARE-1 worst case running time: $\Theta(n^2)$
since there are 2 levels of nested for loops and the lines within the innermost level take constant time

COMPARE-2 worst case running time: $\Theta(n)$
since there is 1 level of for loops and the lines within the for loop take constant time