The exam will be in class (one hour) and closed book. You may not use any additional resources other than the sums and recurrence relations tables and logarithm/exponent rules from class. (These will be provided on the exam.)

It is strongly recommended that you treat this as a practice exam and work on it in a similar environment.

---

1. Showing your work / providing some evidence of where your answers come from can help you earn partial credit if you make a mistake.

   (a) For each of the functions below, find a simple function g such that $f(n) = \Theta(g(n)$ (or $T(n) = \Theta(g(n))$).

   (i) $10 \log^2 n$

   (ii) $5n \log n + 2n^3$

   (iii) 10000

   (iv) $\sum_{i=1}^{n} \left(\frac{2}{3}\right)^i i^2$

   (v) $\sum_{i=1}^{n^2} n \log i$

   (vi) $\sum_{i=1}^{\log n} i$

   (vii) $T(n) = 3T(n/3)+3n$

   (viii) $T(n) = 2T(n-2)+4$

   (ix) $T(n) = T(n/2) + n^2$

   (b) List the functions in order according to growth rate, from slowest growing to fastest growing. If several functions have the same growth rate, indicate that and list them together.

2. For each of the following, if there is a difference between best-case and worst-case behavior, give running times for both. Show your work / provide some evidence of how you arrived at your answers, such as by writing the applicable sums or recurrence relations.

   (a) Give the running time of `alg` for arrays A and B of length n.

```
algorithm alg ( A, B ) :
  input: arrays A, B of length n

 for ( i ← 0 ; i < n ; i += 2 ) do
   B[i] ← A[i]
   B[i+1] ← i

 for ( j ← n-1 ; j ≥ 0 ; j-- ) do
   for ( k ← 1 ; k < j ; k *= 2 ) do
     B[j] ← B[j] + A[k]
```

   (b) Give the running time of `alg` for arrays A and B with lengths m and n, respectively.

```
algorithm alg ( A, m, B, n ) :
  input: array A of length m, array B of length n; m ≤ n

  for ( b ← 0 ; b ≤ n-m ; b++ ) do
    if helper(A,B,b) then
      return b
  return -1


algorithm helper ( A, B, i ) :
  input: array A of length m, array B of length n,
         0 ≤ i ≤ n − m

  for ( a ← 0 ; a < m ; a++ ) do
    if A[a] != B[i+a] then
      return false
  return true
```

(c) Give the running time of `alg` for arrays A, B of length n. You can assume n is a power of 2.

```
algorithm alg ( A, B, n ) :
  input: arrays A, B of length n
  output: an array of length 2n

  if n = 1 then
    return { A[0]*B[0] }
  if n = 2 then
    return { 0, A[1]*B[1], A[1]*B[0]+A[0]*B[1], A[1]*B[1] }

  for ( i ← 0 ; i < n/2 ; i++ ) do
    A'[i] ← A[i]+A[n/2+i]
    B'[i] ← B[i]+B[n/2+i]

  R1 ← alg(helper(A,0,n/2),helper(B,0,n/2),n/2)
  R2 ← alg(helper(A,n/2,n),helper(B,n/2,n),n/2)
  R3 ← alg(A',B',n/2)

  for ( i ← 0 ; i < n ; i++ ) do
    R4 ← R3[i] - R1[i] - R2[i]

  for ( i ← 0 ; i < 2n ; i++ ) do
    R[i] ← 0
    if 0 ≤ i < n then
      R[i] ← R[i] + R1[i]
    if 0 ≤ i-n/2 < n then
      R[i] ← R[i] + R4[i-n/2]
    if 0 ≤ i-n < n then
      R[i] ← R[i] + R2[i-n]
  return R


algorithm helper ( A, i, j ) :
  input: array A, values i < j
  output: an array of length j − i containing A[i..j − 1]

  for ( k ← i ; k < j ; k++ ) do
    B[k-i] ← A[k]
  return B
```

3. In addition to answering the questions (yes or no), briefly explain your answers — why or why not?

   (a) If $f(n) = O(n^2)$, can it also be true that $f(n) = \Theta(n \log n)$?

   (b) If $f(n) = \Omega(n^2)$, can it also be true that $f(n) = O(n \log n)$?

   (c) If an algorithm is described as taking $\Theta(n^2)$ time, is it possible for the best case to be $O(n)$?