This exam is closed book, notes, computer and cell phone.
Show partial solutions to get partial credit.
Make sure you have answered all parts of a question.
If your solutions are not written legibly, you won’t get full credit.
Clarity and succinctness will be rewarded.

Question 1:__________(out of 15)
Question 2:__________(out of 15)
Question 3:__________(out of 10)
Question 4:__________(out of 10)
Question 5:__________(out of 10)
Question 6:__________(out of 10)
Question 7:__________(out of 10)
Question 8:__________(out of 10)
Question 9:__________(out of 10)

Total:__________(out of 100)
1. Short questions:

(a) Why is Strassen’s Matrix Multiplication more important for fast computers?

*With fast computers you can solve large problems (i.e. large n). The larger the problem, the more the constants don’t matter and the order of growth of the time bound becomes important.*

(b) What is the running time of CountSort?

Explain all your variables!

\[ O(n + k), \]

\( n \) is number of keys

in the range 0..\( k – 1 \)

(c) What is a disadvantage of MergeSort?

Requires \( \Omega(n) \) additional space.
- high constant in the average case

(d) Which sorting algorithm is fast when the array is almost sorted?

What is its worst-case running time?

**InsertionSort:** \( \Theta(n^2) \)

(e) What is the minimum number of comparisons it takes to find the maximum of \( n \) keys?

\( n – 1 \)
2. Show by induction that the number of nodes in a complete binary tree of height $h$ is $2^{h+1} - 1$.

Prove this by an induction on the height of the tree.

Definitions: A binary tree is a tree where each node as 0 or 2 children.

In a complete binary tree every level has the maximum number of nodes.

The height is the maximum number of edges on a path from the root to a leaf.

Hints:

- Draw some small complete binary trees with heights 0, 1, 2, 3.
- Weak or strong induction?
- What is your base case?

Proof A)

Induction on the height of the tree $h = 0, 1, \ldots$.

Base case: $h = 0$.
In this case the tree consists of a single node and $2^{0+1} - 1 = 2 - 1 = 1$.

Induction hypothesis:
A complete binary tree of height $h$ has $2^{h+1} - 1$ nodes.

Given a complete binary tree of height $h + 1$. Remove the root. This splits tree into the root, plus two complete binary trees of height $h$ with $2^{h+1} - 1$ nodes each, for a total # of nodes of:

$$2 (2^{h+1} - 1) + 1 = 2^{h+2} - 1.$$ 

Proof B)

Setup as above but show that: A complete binary tree of height $h$ has $2^{h+1} - 1$ nodes and (include this into the statement to prove) the bottom layer has $2^h$ nodes.

Base case is the same. Clearly bottom layer has $2^0 = 1$ nodes.

Given a tree of height $h + 1$. Remove the bottom layer, leaving a complete binary tree of height $h$. By the inductive assumption it has $2^{h+1} - 1$ nodes, $2^h$ of which are on the bottom layer. Now add the bottom layer back in. Clearly this bottom layer has twice as many nodes as the bottom layer of the height $h$ tree, i.e. $22^h = 2^{h+1}$ nodes and the total number of nodes is

$$2^{h+1} - 1 + 2^{h+1} = 2^{h+2} - 1.$$
3. (a) What is the worst-case running time of QuickSort?

\[ \Theta(n^2) \]

(b) What is the recurrence on the number of comparisons in this case?

\[ T(n) = T(n - 1) + n - 1 \]

(c) When does this case happen?

Keys are sorted in ascending or descending order.

(d) How can it be avoided (or at least made very unlikely)?

Pick pivots at random or randomly permute the initial sequence of keys. Now the average case is always \( \theta(n \log n) \) with a small constant (assuming you have good random # generator).
4. Consider the following heap

(a) Redraw the heap after inserting the key 5.

(b) Redraw the heap after doing a Deletemax operation to the heap on top of the page.

(c) Name two operations besides ExtractMax and Insert that can be done efficiently on a heap?

**ChangeKey and DeleteMax**

What are the running times of each of the two operations?

**Both or \( \Theta(\log n) \).**
5. Show that \( \sum_{i=1}^{n} 2 \log(i + 1) \) is \( \Theta(n \log n) \).

\[
\sum_{i=1}^{n} 2 \log(i + 1) \leq \sum_{i=1}^{n} 2 \log(n + 1)
= 2n \log(n + 1)
\leq 2n \log(2n)
= 2n \log n + 2n \log 2
= 2n \log n + O(n)
= O(n \log n)
\]

\[
\sum_{i=1}^{n} 2 \log(i + 1) \geq \sum_{i=1}^{n} 2 \log i
\geq \sum_{i=n/2}^{n} 2 \log n / 2
= n / 2 \log n / 2
= n \log n + n \log 1 / 2
\geq \max(n \log n, n \log 1 / 2)
= n \log n, \text{ for large enough } n
= \Omega(n \log n)
\]
6. Use the below Master Theorem to give a tight asymptotic bound for the following recurrence:

\[ T(n) = 2T(n/2) + 10n. \]

\[ a = 2, \quad b = 2, \quad \log_2 2 = 1 \]

This means we can use case 2, since \( f(n) = 10n = O(n^1) \).

We conclude that \( T(n) = \Theta(n \log n) \).

---

**Master Theorem.** Let \( a \geq 1 \) and \( b > 1 \) be constants, let \( f(n) \) be a function, and let \( T(n) \) be defined on the nonnegative integers by the recurrence

\[ T(n) = aT(n/b) + f(n). \]

Then \( T(n) \) has the following asymptotic bounds:

(a) If \( f(n) = O(n^{\log_b a - \varepsilon}) \) for some constant \( \varepsilon > 0 \), then \( T(n) = \Theta(n^{\log_b a}) \).

(b) If \( f(n) = O(n^{\log_b a}) \), then \( T(n) = \Theta(n^{\log_b a \log \log n}) \).

(c) If \( f(n) = \Omega(n^{\log_b a + \varepsilon}) \), for some constant \( \varepsilon > 0 \),
   and if \( af(n/b) \leq cf(n) \) for some constant \( c < 1 \) and all sufficiently large \( n \),
   then \( T(n) = \Theta(f(n)) \).
7. You are given an $n$ distinct keys.
   After a preprocessing step you should be able to compute queries of the following form in $O(1)$ time:
   
   Input: $1 \leq i \leq j \leq n$
   
   Output: the sum of the $i$th largest,
   plus the $(i + 1)$st largest,
   plus the $(i + 2)$nd largest, up until the $j$th largest.

   (a) What is the preprocessing you need to do?
       How expensive is it?

       Sort the keys in descending order into array $A$.
       This costs $\Theta(n \log n)$ w.c. time.
       Compute the initial totals in an additional $O(n)$ time:

       $S(0) := 0$
       for $i = 1..n$
       $S(i) := S(i - 1) + A(i)$

       Total time of preprocessing is $\Theta(n \log n)$.

   (b) With the preprocessing, how do you process a query?
       What is the running time?

       If $j < i$ then output 0
       otherwise output the difference $S(j) - S(i - 1)$. 
8. (a) What is the information theoretic lower bound on the number of comparisons needed for the following problem:

*Find the largest and smallest key out of a set of $n$ keys?*

\[
\lg(\text{“number of outcomes”}) = \lg(n(n - 1)) = \Theta(\lg n).
\]

This is weak bound.

(b) Sketch and algorithm solving this problem with as few comparisons as possible.

For the sake of simplicity assume $n$ is even. Compare pairs of numbers until you run out.

Look for max among the winners

and the min among the losers.

The number of comparisons is $\frac{n}{2}$ for the first and $\frac{n}{2} - 1$ for each of the last two steps.

The total # is $\frac{3n}{2} - 2$.

In general $\left\lceil \frac{3n}{2} \right\rceil - 2$ comparisons are required.
9. Show that \((n + 1) \ln n\) is \(o(n^2)\).

Use l’Hospital’s rule:

\[
\lim_{n \to \infty} \frac{(n + 1) \ln n}{n^2} = \lim_{n \to \infty} \frac{\ln n + \frac{n+1}{n}}{2n}
\]

\[
= \lim_{n \to \infty} \frac{\ln n + 1 + \frac{1}{n}}{2n}
\]

\[
= \lim_{n \to \infty} \frac{\frac{1}{n}}{\frac{1}{n^2}}
\]

\[
= 0
\]