Problem 1, Analysis of Algorithm (10 points)
Where n is a power or 2 and $n > 8$, how many x's does the function Mystery(n) below print?
(a) Write and solve the recurrence exactly using substitution.
(b) State the solution in $O()$ notation.

```
Mystery(n)
if n > 8 then begin
        print("x")
        Mystery(n/2)
        Mystery(n/2)
        end
```

Answer: We have to solve the recurrence $T(n) = 2T\left(\frac{n}{2}\right) + 1$ for $n > 8$, with $T(8) = 0$.

$$\begin{aligned}
T(n) &= 2T\left(\frac{n}{2}\right) + 1 \\
&= 2\left(2T\left(\frac{n}{2^2} + 1\right)\right) + 1 = 2^2 T\left(\frac{n}{2^2}\right) + 2 + 1 \\
&= 2^2\left(2T\left(\frac{n}{2^3} + 1\right)\right) + 2 + 1 = 2^3 T\left(\frac{n}{2^3}\right) + 2^2 + 2 + 1 \\
&\quad \text{guessing general form} \\
&= 2^k T\left(\frac{n}{2^k}\right) + 2^{k-1} + \ldots + 2^2 + 2 + 1 \\
&= 2^k T\left(\frac{n}{2^k}\right) + 2^k - 1 \\
&\quad \text{know that } T(8) = 0, \text{ so stop when } \tfrac{n}{2^k} = 8, \text{ equivalently } k = \log_2 n - 3 \\
&= 2^k T(8) + 2^{\log_2 n - 3} - 1 \\
&= 2^{-3} \times 2^{\log_2 n} - 1 \\
&= \frac{1}{8} \times n - 1 \\
&= O(n)
\end{aligned}$$

PS. Sanity test for solution: $T(n) = \frac{1}{8}n - 1$.
For $n = 8$ we need to check if $T(8) = 0$. Indeed $T(8) = \frac{1}{8}8 - 1 = 0$.
For $n = 16$ we need to check if $T(16) = 1$. Indeed $T(16) = \frac{1}{8}16 - 1 = 2 - 1 = 1$.

Problem 2, Analysis of Algorithm (10 points)
Where n is a power or 2 and $n \geq 1$, how many x's does the function Mystery(n) below print?
(a) Write and solve the recurrence exactly using substitution.
(b) State the solution in $O()$ notation.

```
Mystery(n)
print("xx")
if n > 1 then begin
        Mystery(n/2)
        Mystery(n/2)
        Mystery(n/2)
        end
```

Answer: We have to solve the recurrence $T(n) = 3T\left(\frac{n}{2}\right) + 2$ for $n > 1$, with $T(1) = 2$.

$$
\begin{aligned}
T(n) &= 3T\left(\frac{n}{2}\right) + 2 \\
&= 3\left(3T\left(\frac{n}{2^2}\right) + 2\right) + 2 = 3^2 T\left(\frac{n}{2^2}\right) + 3 \times 2 + 2 \\
&= 3^2\left(3T\left(\frac{n}{2^3} + 2\right)\right) + 3 \times 2 + 2 = 3^3 T\left(\frac{n}{2^3}\right) + 3^2 \times 2 + 3 \times 2 + 2 \\
&\quad \text{guessing general form} \\
&= 3^k T\left(\frac{n}{2^k}\right) + 2 \times (3^{k-1} + \ldots + 3^2 + 3 + 1) \\
&= 3^k T\left(\frac{n}{2^k}\right) + 2\left(\frac{3^k - 1}{3 - 1}\right) \\
&= 3^k T\left(\frac{n}{2^k}\right) + 3^k - 1 \\
&\quad \text{we know that } T(2^0) = T(1) = 2, \text{ so stop when } \tfrac{n}{2^k} = 1, \text{ equivalently } k = \log_2 n \\
&= 3^{\log_2 n} T(1) + 3^{\log_2 n} - 1 \\
&= 3^{\log_2 n} \times 2 + 3^{\log_2 n} - 1 \\
&= 3 \times 3^{\log_2 n} - 1 \\
&= 3 \times 2^{\log_2 3 \log_2 n} - 1 \\
&= 3n^{\log_2 3} - 1 \\
&= O(n^{\log_2 3})
\end{aligned}
$$

PS. Sanity test for solution: $T(n) = 3n^{\log_2 3} - 1$.
For $n = 1$ we need to check if $T(1) = 2$. Indeed $T(1) = 3 \times 1^{\log_2 3} - 1 = 3 - 1 = 2$.
For $n = 2$ we need to check if $T(2) = 8$. Indeed $T(2) = 3 \times 2^{\log_2 3} - 1 = 3 \times 3 - 1 = 9 - 1 = 8$.

Problem 3, Analysis of Algorithm (10 points)
Where n is a power or 2 and $n \geq 1$, how many x's does the function Mystery(n) below print?
(a) Write and solve the recurrence exactly using substitution.
(b) State the solution in $O()$ notation.

```
Mystery(n)
for i := 1 to n^2
        print("xx")
if n > 1 then begin
                Mystery(n/2)
                Mystery(n/2)
                Mystery(n/2)
                end
```

Answer: We have to solve the recurrence $T(n) = 3T\left(\frac{n}{2}\right) + 2n^2$ for $n > 1$, with $T(1) = 2$.

$$
\begin{aligned}
T(n) &= 3T\left(\frac{n}{2}\right) + 2n^2 \\
&= 3\left(3T\left(\frac{n}{2^2}\right) + 2\left(\frac{n}{2}\right)^2\right) + 2n^2 = 3^2T\left(\frac{n}{2^2}\right) + 2n^2\left(\frac{3}{2^2}\right) + 2n^2 \\
&= 3^2\left(3T\left(\frac{n}{2^3}\right) + 2\left(\frac{n}{2^2}\right)^2\right) + 2n^2\left(\frac{3}{2^2}\right) + 2n^2 = 3^3T\left(\frac{n}{2^3}\right) + 2n^2 + \left(\frac{3}{2^2}\right)^2 + 2n^2\left(\frac{3}{2^2}\right) + 2n^2 \\
&\quad \text{guessing general form} \\
&= 3^kT\left(\frac{n}{2^k}\right) + 2n^2\left(\left(\frac{3}{2^2}\right)^{k-1} + \ldots + \left(\frac{3}{2^2}\right)^2 + \left(\frac{3}{2^2}\right) + 1\right) \\
&= 3^kT\left(\frac{n}{2^k}\right) + 2n^2\left(\frac{1 - \left(\frac{3}{2^2}\right)^k}{1 - \frac{3}{4}}\right) \\
&= 3^kT\left(\frac{n}{2^k}\right) + 8n^2\left(1 - \left(\frac{3}{2^2}\right)^k\right) \\
&\quad \text{we know that } T(1) = 2, \text{ so stop when } \tfrac{n}{2^k} = 1, \text{ equivalently } k = \log_2 n \\
&= 3^{\log_2 n}T(1) + 8n^2\left(1 - \left(\frac{3}{2^2}\right)^{\log_2 n}\right) \\
&= 3^{\log_2 n} \times 2 + 8n^2 - 8n^2\frac{3^{\log_2 n}}{2^{2\log_2 n}} \\
&= 2n^{\log_2 3} + 8n^2 - 8n^2 n^{\log_2 3}\frac{1}{n^2} \\
&= 8n^2 - 6n^{\log_2 3} \\
&= O(n^2)
\end{aligned}
$$

PS. Sanity test for solution: $T(n) = 8n^2 - 6n^{\log_2 3}$.
For $n = 1$ we need to check if $T(1) = 2$. Indeed $T(1) = 8 \times 1 - 6 \times 1 = 2$.
For $n = 2$ we need to check if $T(2) = 14$. Indeed $T(2) = 8 \times 2^2 - 6 \times 2^{\log_2 3} = 32 - 18 = 14$.

Problem 4, Analysis of Algorithm (10 points)
Where n is a positive integer, how many x's does the function Mystery(n) below print?
(a) Write and solve the recurrence exactly using substitution.
(b) State the solution in $O()$ notation.

```
Mystery(n)
if n = 1 then print("x")
if n > 1 then begin
              Mystery(n − 1)
              Mystery(n − 1)
              end
```

Answer: We need to solve the recurrence $T(n) = 2T(n-1)$, with $T(1) = 1$.

$$\begin{aligned} T(n) &= 2T(n-1) \\ &= 2^2T(n-2) \\ &= 2^3T(n-3) \\ &= \text{general form} \\ &= 2^kT(n-k) \\ &\quad \text{we know that } T(1) = 1, \text{ so stop when } n-k = 1, \text{ equivalently } k = n-1 \\ &= 2^{n-1}T(1) \\ &= 2^{n-1} \\ &= O(2^n) \end{aligned}$$

PS. Sanity test for solution: $T(n) = 2^{n-1}$.
For $n = 1$ we need to check if $T(1) = 1$. Indeed $T(1) = 2^{1-1} = 2^0 = 1$.
For $n = 2$ we need to check if $T(2) = 2$. Indeed $T(2) = 2^{2-1} = 2^1 = 2$.
For $n = 3$ we need to check if $T(3) = 4$. Indeed $T(3) = 2^{3-1} = 2^2 = 4$.

Problem 5, Min and Max with Fewer Comparisons (10 points)
Let $a_1 \dots a_n$ be an input array of n unsorted distint integers, where n is an even number. Let $m_{\max}$ be the maximum value of the above integers, and let $m_{\min}$ be the minimum value of the above integers. Give an algorithm that findsboth $m_{\max}$ and $m_{\min}$ using at most $\frac{3n}{2}-2$ comparisons.
Argue correctness and running time of your solution.

Answer:
Using $\frac{n}{2}$ comparisons, one for each pair (a_i, a_{2i}), $i \in \{1, \dots, \frac{n}{2}\}$, we idenitify two groups:
The first group consists of $\frac{n}{2}$ distinct integers that "won" in the paiwise comparisons, and thus this group contains the maximum of the entire sequence $m_{\max}$. Now with $\left(\frac{n}{2}-1\right)$ amonf the elements of the first group we can find $m_{\max}$.
The second group consists of $\frac{n}{2}$ distinct integers that "lost" in the paiwise comparisons, and thus this group contains the minimum of the entire sequence $m_{\min}$. Now with $\left(\frac{n}{2}-1\right)$ amonf the elements of the first group we can find $m_{\min}$.
This the total number of comparisons is $\frac{n}{2} + 2 \times \left(\frac{n}{2}-1\right) = \frac{3n}{2}-2$.

Problem 6, Sorting Faster than $O(n \log n)$ in Special Case (10 points)
Let $a_1 \dots a_n$ be an input array of n unsorted and not necessarily distint integers. Let $m_{\max}$ be the maximum value of the above integers, and let $m_{\min}$ be the minimum value of the above integers. Let $M = m_{\max} - m_{\min}$. Give an $O(n + M)$ comparison algorithm that sorts the input array.
Argue correctness and running time of your solution.

Answer:
STEP 1: Using $2(n-1)$ comparisons we fing the values $m_{\max}$ and $m_{\min}$.
STEP 2: We compute $M = m_{\max} - m_{\min}$.
We now know that the total number of distinct values that appear is $M+1$.
STEP 3: We create a new array $m_0, m_1, \dots, m_M$ and initialize it to zero:
for $i := 0$ to M set $m_i := 0$
STEP 4: m_i will eventually hold the number of elements of the initial array that are equal to $(m_{\min}+i)$.
for $j := 1$ to n
if $((a_j - m_{\min}) = i)$ then $(m_i := m_i + 1)$
STEP 5: For $0 \le i \le M$, Output: m_i copies of the value $i + m_{\min}$:
$j := 0$
for $i := 0$ to M

$$\text{while } m_i > 0 \begin{cases} j := j + 1 \\ a_j := m_{\min} + i \\ m_i := m_i - 1 \end{cases}$$

Complexity Analysis:
STEP 1: $O(n)$
STEP 2: constant
STEP 3: $O(M)$
STEP 4: One scanning of original array, thus $O(n)$
STEP 5: $O(n + M)$
Total: $O(n + M)$

Problem 7, Use of Pointers (10 points)
Let $a_1 \dots a_n$ be n sorted distint integers, and let τ be an additional given integer. Give an $O(n)$ comparison algorithm that decides if there exist distinct indices i and j, ie $1 \leq i < j \leq n$, such that $a_i + a_j = \tau$.
Argue correctness and running time of your solution.

Answer:
Main Fact: (1) If $a_1 + a_n = \tau$ then we have a solution.
(2) If $a_1 + a_n < \tau$ then a_1 is excluded from any solution (because all the other a_i's are $< a_n$.)
(3) If $a_1 + a_n > \tau$ then a_n is expluded from any solution (because all the other a_i's are $> a_1$.)

The above main fact suggests a linear algorithm which keeps two pointers LEFT and RIGHT.
Initially LEFT := 1 and RIGHT := n, and in each step, either a solution is found, or the LEFT pointer moves one step to the right, or the right pointer moves one step to the left. This repeats at most until the pointers meet, in which case there is no solution, because we have excluded all elements of the array.

Problem 8, Searching a Tree (10 points)
You are given a complete binary tree on n nodes, where each node has a distinct value w_i, $1 \leq i \leq n$.
The input representation is as follows:
(1) Index 1 is the root of the tree.
(2) For $1 \leq i \leq \frac{n-1}{2}$, the left child of i is $2i$ and the right child of i is $2i+1$.
(3) For $2 \leq i \leq n$, the parent of i is $\lfloor \frac{n}{2} \rfloor$.
Say that k is a *local minimum* if and only if:
(1) If $k = 1$, then w_1 is smaller than both its children.
(2) If $k \geq \frac{n-1}{2}$, then w_k is smaller than its parent.
(3) If $2 \leq k \leq \frac{n-1}{2}$, then w_k is smaller than both its children, and w_k is also smaller than its parent.
Give an $O(\log n)$ comparison algorithm that finds a local minimum of the binary tree.
Justify correctness and running time.

Answer:
Main Fact: Every binary tree always has at least one local minimum, namely the minimum of all the values involved in the tree.

If the root is smaller than both its children, then the root is a local minimum and we are done.
If the root is larger than at least one of its children, say larger than its left child, then (a)if the left child is smaller than both of its children we have found a local minimum, (b)if the left child is larger than at least one of its children, then we recurse looking for a local minimum in the complete binary tree rooted at this left child.
(Note that if we reach a leaf that is smaller than its parent, then this is immediately a local minimum, and we terminate.)
This process may repeat at most $\log_2 n$ times which is the depth of a complete binaty tree.
And at each level we need to do at most 2 comparisons.
So the total complexity is $O(\log_2 n)$.

Problem 9, Sorting in Linear Time in Special Case (10 points)
Suppose that n is a perfect square and let $N = n + \sqrt{n}$.
You are given an array of $a_1 \dots a_N$ distinct integers, where the first n integers are sorted $a_1 < a_2 < \dots < a_n$, but the last $\sqrt{n}$ integers are not sorted. Give an $O(N)$ comparison algorithm that sorts the entire input array $a_1 \dots a_N$.

Answer:
Using any nontrivial sorting algorithm, eg Bubblesort, we sort the last $\sqrt{n}$ elements in $O(\sqrt{n}^2) = O(n)$.
We now have two sorted lists, the first one of length n and the second one of length $\sqrt{n}$, which we can merge to one sorted list using $O(N)$ comparisons.

Problem 10, Comparing Algorithmic Performance (10 points)
Suppose you are choosing between the following three algorithms:
• Algorithm A solves problems by dividing them into five subproblems of half the size, recursively solving each subproblem, and then combining solutions in linear time.
• Algorithm B solves problems of size n by recursively solving two subproblems of size $(n-1)$ and then combining the solutions in constant time.
• Algorithm C solves problems of size n by dividing them into nine subproblems of size $n/3$, recursively solving each subproblem, and then combining the solutions in $O(n^2)$ time.
Solve each recurrence by substitution, and give the running times of each of these algorithms in $O()$ notation. Which one would you choose as the asymptotically fastest?

Answer:
• For Algorithm A we have to solve the recurrence $T(n) = 5T\left(\frac{n}{2}\right) + cn$ for all $n > n_0$, where n_0 is a positive integer. The base case is $T(n_0) = c_0$, where c_0 is also a positive integer. Of course, n_0 and c_0 are constants, independent of n. We will solve this recurrence by substitution, and we will assume, without loss of generality, that n and n_0 are powers of 2.

$$
\begin{aligned}
T(n) &= 5T\left(\frac{n}{2}\right) + cn \\
&= 5\left(5T\left(\frac{n}{2^2}\right) + c\frac{n}{2}\right) + cn = 5^2T\left(\frac{n}{2^2}\right) + c\frac{n}{2} + cn \\
&= 5^2\left(5T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2}\right) + c\frac{n}{2} + cn = 5^3T\left(\frac{n}{2^3}\right) + c\frac{n}{2^2} + c\frac{n}{2} + cn \\
&\quad \text{guessing general form} \\
&= 5^kT\left(\frac{n}{2^k}\right) + c\frac{n}{2^{k-1}} + \ldots + c\frac{n}{2} + cn \\
&= 5^kT\left(\frac{n}{2^k}\right) + cn\left(\frac{1}{2^{k-1}} + \ldots + \frac{1}{2} + 1\right) \\
&= 5^kT\left(\frac{n}{2^k}\right) + cn\left(\frac{1-\frac{1}{2^k}}{1-\frac{1}{2}}\right) \\
&= 5^kT\left(\frac{n}{2^k}\right) + 2cn\left(1 - \frac{1}{2^k}\right) \\
&\quad \text{know } T(n_0) = c_0\text{, so stop when } \tfrac{n}{2^k} = n_0\text{, equivalently } k = \log_2 n - \log_2 n_0 \\
&= 5^{\log_2 n - \log_2 n_0}T\left(\frac{n}{2^{\log_2 n - \log_2 n_0}}\right) + 2cn\left(1 - \frac{1}{2^{\log_2 n - \log_2 n_0}}\right) \\
&= 5^{\log_2 n}5^{-\log_2 n_0}T\left(\frac{n}{2^{\log_2 n}2^{-\log_2 n_0}}\right) + 2cn\left(1 - \frac{1}{2^{\log_2 n}2^{-\log_2 n_0}}\right) \\
&= \frac{1}{n_0^{\log_2 5}} \times n^{\log_2 5} \times T(n_0) + 2cn\left(1 - \frac{n_0}{n}\right) \\
&= n^{\log_2 5}\frac{c_0}{n_0^{\log_2 5}} + 2cn - 2cn_0 \qquad \text{Note: Leading term is } n^{\log_2 5} \\
&= O(n^{\log_2 5})
\end{aligned}
$$

PS. Sanity test for solution: $T(n) = n^{\log_2 5}\frac{T(n_0)}{n_0^{\log_2 5}} + 2cn - 2cn_0$.
For $n = n_0$ we need to check if $T(n_0) = c_0$.
Indeed: $T(n_0) = {n_0}^{\log_2 5}\frac{c_0}{n_0^{\log_2 5}} + 2cn_0 - 2cn_0 = c_0$.

• For Algorithm B we have to solve the recurrence $T(n) = 2T\,(n-1)+c$ for all $n > n_0$, where n_0 is a positive integer. The base case is $T(n_0) = c_0$, where c_0 is also a positive integer. Of course, n_0 and c_0 are constants, independent of n.

$$
\begin{aligned}
T(n) &= 2T\,(n-1) + c \\
&= 2\,(2T(n-1) + c) + c = 2^2 T\,(n-2) + 2c + c \\
&= 2^2\,(2T(n-3) + c) + 2c + c = 2^3 T(n-3) + 2^2 c + 2c + c \\
&\quad \text{gessing general form} \\
&= 2^k T(n-k) + c(2^{k-1} + \ldots + 2^2 + 2 + 1) \\
&= 2^k T(n-k) + c(2^k - 1) \\
&\quad \text{know } T(n_0) = c_0\text{, so stop when } n-k = n_0\text{, equivalently } k = n - n_0 \\
&= 2^n \times \frac{1}{2^{n_0}} \times T(n_0) + c\left(\frac{2^n}{2^{n_0}} - 1\right) \\
&= 2^n \times \frac{1}{2^{n_0}} \times c_0 + 2^n \frac{c}{2^{n_0}} - c \\
&\quad \text{Note: Leading term is } 2^n \\
&= O(2^n)
\end{aligned}
$$

PS. Sanity test for solution: $T(n) = 2^n \times \frac{1}{2^{n_0}} \times c_0 + 2^n \frac{c}{2^{n_0}} - c$.
For $n = n_0$ we need to check if $T(n_0) = c_0$.
Indeed: $T(n_0) = 2^{n_0} \times \frac{1}{2^{n_0}} \times c_0 + 2^{n_0} \frac{c}{2^{n_0}} - c = c_0 + c - c = c_0$.

• For Algorithm C we have to solve the recurrence $T(n) = 9T\left(\frac{n}{3}\right) + cn^2$ for all $n > n_0$, where n_0 is a positive integer. The base case is $T(n_0) = c_0$, where c_0 is also a positive integer. Of course, n_0 and c_0 are constants, independent of n. We will solve this recurrence by substitution, and we will assume, without loss of generality, that n and n_0 are powers of 3.

$$
\begin{aligned}
T(n) &= 9T\left(\frac{n}{3}\right) + cn^2 \\
&= 9\left(9T\left(\frac{n}{3^2}\right) + c\left(\frac{n}{3}\right)^2\right) + cn^2 = 9^2T\left(\frac{n}{3^2}\right) + 2cn^2 \\
&= 9^2\left(9T\left(\frac{n}{3^3}\right) + c\left(\frac{n}{3^2}\right)^2\right) + 2cn^2 = 9^3T\left(\frac{n}{3^3}\right) + 3cn^2 \\
&= 9^3\left(9T\left(\frac{n}{3^4}\right) + c\left(\frac{n}{3^3}\right)^2\right) + 3cn^2 = 9^4T\left(\frac{n}{3^4}\right) + 4cn^2 \\
&\quad \text{guessing general form} \\
&= 9^kT\left(\frac{n}{3^k}\right) + kcn^2 \\
&\quad \text{know } T(n_0) = c_0\text{, so stop when } \tfrac{n}{3^k} = n_0\text{, equivalently } k = \log_3 n - \log_3 n_0 \\
&= 9^{\log_3 n} \times \frac{1}{9^{\log_3 n_0}} \times T(n_0) + cn^2\log_3 n - cn^2\log_3 n_0 \\
&= 3^{2\log_3 n} \times \frac{1}{3^{2\log_3 n_0}} \times c_0 + cn^2\log_3 n - cn^2\log_3 n_0 \\
&= n^2\frac{c_0}{n_0^2} + cn^2\log_3 n - cn^2\log_3 n_0 \\
&\quad \text{The leading term is } n^2\log_3 n \\
&= O(n^2\log_3 n)
\end{aligned}
$$

PS. Sanity test for solution: $T(n) = n^2\frac{c_0}{n_0^2} + cn^2\log_3 n - cn^2\log_3 n_0$.

For $n = n_0$ we need to check if $T(n_0) = c_0$.

Indeed: $T(n_0) = n_0^2\frac{c_0}{n_0^2} + cn_0^2\log_3 n_0 - cn_0^2\log_3 n_0 = c_0$.

FINAL ANSWER:

Algorithm B is exponential, so it is asympotically the slowest.

Algorithm C which is $O(n^2\log_3 n)$ is asymptotically faster than algorithm A which is $O(n^{\log_2 5})$.

To see this, realise that $n^{\log_2 5} = n^{\log_2\left(5\times\frac{4}{4}\right)} = n^{\log_2\left(4\times\frac{5}{4}\right)} = n^{\log_2 4}n^{\log_2\frac{5}{4}} = n^2n^{\log_2\frac{5}{4}}$.

So the comparison is really between $\log_3 n$ and $n^{\log_2\frac{5}{4}}$.

And any polylogarithmic function runs asymptotically faster than any n^ϵ, for any $\epsilon > 0$.