1. Suppose you have 3 algorithms $A_1$, $A_2$, $A_3$ all for the same problem. Assume, all input can be divided into 4 groups, say, $X_1$, $X_2$, $X_3$, $X_4$ and you happen to know performance, in terms of running time, of these algorithms over all possible inputs of size $n$. for each group,

<table>
<thead>
<tr>
<th></th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$5n^3 + n^{1/2}$</td>
<td>$100n^4 + n \log n$</td>
<td>$0.1n^2 + 2^n$</td>
<td>$n^5 + 10\log n$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$10n^2$</td>
<td>$0.0001n^2$</td>
<td>$n^2 + 900n + 5n \log n$</td>
<td>$500n^2 + 100n \log n$</td>
</tr>
<tr>
<td>$A_3$</td>
<td>$0.05n^3 \log n$</td>
<td>$100n^3 \log n$</td>
<td>$125n^3 \log n + n^3 \log n$</td>
<td>$n \log n + 0.01n^5$</td>
</tr>
</tbody>
</table>

Give performance of $A_1$, $A_2$, $A_3$ in terms of $O()$, $\Omega()$, $\Theta()$ notation. **6 pts**

<table>
<thead>
<tr>
<th></th>
<th>$X_1$</th>
<th>$X_2$</th>
<th>$X_3$</th>
<th>$X_4$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$n^3$</td>
<td>$n^4$</td>
<td>$2^n$</td>
<td>$n^5$</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>$A_3$</td>
<td>$n^3 \log n$</td>
<td>$n^4 \log n$</td>
<td>$n^3 \log n$</td>
<td>$n^3$</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>$O()$</th>
<th>$\Omega()$</th>
<th>$\Theta()$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A_1$</td>
<td>$2^n$</td>
<td>$n^3$</td>
<td>--</td>
</tr>
<tr>
<td>$A_2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
<td>$n^2$</td>
</tr>
<tr>
<td>$A_3$</td>
<td>$n^3 \log n$</td>
<td>$n^4$</td>
<td>--</td>
</tr>
</tbody>
</table>
2. Assume $X$ is an algebraic object whose multiplication is relatively expensive. We want to compute $X^n$ for various $n$.

- **A**
  Assume you can store as many $X^j$ as you wish, how would you compute $X^n$ for $n=101, 300, 544$.
  Indicate which $X^j$ you store and how do you compute desired $X^n$’s? (Show which multiplications)

  6 pts

  $n = 101 = 64 + 32 + 4 + 1 = 2^6 + 2^5 + 2^2 + 2^0 \implies X^{101} = X^{64} \ast X^{32} \ast X^4 \ast X$. Thus 6+3 multiplication is needed.

  $n = 300 = 256 + 32 + 8 + 4 \implies X^{300} = X^{256} \ast X^{32} \ast X^8 \ast X^4$. Thus $8 + 3 = 11$

  $n = 544 = 512 + 32 \implies X^{544} = X^{512} \ast X^{32}$. Thus $9 + 1 = 10$ mult.

- **B** Given $b(0), b(1), b(k)$, binary expansion of $n$, with $b(k)=1$, how would you compute $X^n$ without storing all intermediate results. Write an algorithm/pseudo-code!

  4 pts

  Let $P$ be product $X^n$, and $T$ be the term $X^{2^i}$. The following algorithm will compute $X^n$

  
  ```plaintext
  P=1 T=X
  for i=0 to k
  if b(i)=1 then P=P \ast T fi
  T = T \ast T
  endfor
  ```
3. Fibonacci

Given \( F(n + 2) = F(n + 1) + F(n) \) with \( F(0) = 0, F(1) = 1 \) you can compute:

a) as a recursion function,

b) as an array using addition,

c) using matrix multiplication via

\[
A^1 = A = \begin{bmatrix} 1 & 1 \\ 1 & 0 \end{bmatrix} \quad A^n = \begin{bmatrix} F(n + 1) & F(n) \\ F(n) & F(n - 1) \end{bmatrix}
\]

Assume multiplication of two 2x2 matrix costs \( \alpha \) additions.

- Assume \( n = 2^k \) and \( \alpha = 2^t \). Compute cost of \( F(n) \) using b) and c) in terms of additions. Write an equation/inequality to determine when matrix multiplication becomes cheaper.

**Solution:** \( k \alpha \leq n = 2^k \Rightarrow k2^t \leq 2^k \)

- For \( k = 2^l \) determine first \( l \) such that matrix multiplication is cheaper for \( t=10 \)

**Solution.** \( k = 2^l \rightarrow 2^l \cdot 2^t \leq 2^{2t} \). Clearly for \( l=1,2,3 \) this equation does not hold. But for \( l=4 \), we have

\[
2^4 \cdot 2^{10} = 2^{14} \leq 2^{16} \quad \text{which is true}
\]

Thus first such \( l \) is \( l=4 \).

- For \( \alpha = 400 \) compute cost of computing \( F(n) \) for \( n=1536 \) using b) and c)

**Solution:** Since \( n=1536=1024+512 \) and \( \alpha = 400 \), clearly we calculate \( F(n) \) via b) in 1536 additions, If we compute via c) we need 10+1 matrix multiplication involving equivalently, 11*400= 4400 additions.
4. **merge sort.** Given the following list, apply merge sort. Show, in each stage sorted elements in a box. First row is unsorted list. Second row is the first stage indicating each sublist of a single element is sorted.  

<table>
<thead>
<tr>
<th></th>
<th>10</th>
<th>3</th>
<th>34</th>
<th>25</th>
<th>6</th>
<th>2</th>
<th>30</th>
<th>47</th>
<th>9</th>
<th>5</th>
<th>7</th>
<th>24</th>
<th>18</th>
<th>11</th>
<th>39</th>
<th>28</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>10</td>
<td>3</td>
<td>34</td>
<td>25</td>
<td>6</td>
<td>2</td>
<td>30</td>
<td>47</td>
<td>9</td>
<td>5</td>
<td>7</td>
<td>24</td>
<td>18</td>
<td>11</td>
<td>39</td>
<td>28</td>
</tr>
<tr>
<td>2nd</td>
<td>3</td>
<td>10</td>
<td>25</td>
<td>34</td>
<td>2</td>
<td>6</td>
<td>30</td>
<td>47</td>
<td>5</td>
<td>9</td>
<td>7</td>
<td>24</td>
<td>11</td>
<td>18</td>
<td>28</td>
<td>29</td>
</tr>
<tr>
<td>3rd</td>
<td>3</td>
<td>10</td>
<td>25</td>
<td>34</td>
<td>2</td>
<td>6</td>
<td>30</td>
<td>47</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>24</td>
<td>11</td>
<td>18</td>
<td>28</td>
<td>39</td>
</tr>
<tr>
<td>4th</td>
<td>2</td>
<td>3</td>
<td>6</td>
<td>10</td>
<td>25</td>
<td>30</td>
<td>34</td>
<td>47</td>
<td>5</td>
<td>7</td>
<td>9</td>
<td>11</td>
<td>18</td>
<td>24</td>
<td>28</td>
<td>39</td>
</tr>
<tr>
<td>5th</td>
<td>2</td>
<td>3</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>b9n</td>
<td>10</td>
<td>11</td>
<td>18</td>
<td>24</td>
<td>25</td>
<td>28</td>
<td>30</td>
<td>34</td>
<td>39</td>
<td>47</td>
</tr>
</tbody>
</table>
5. We want to sort the elements 21 10 30 5 25 35 8 11 15 28 20 40 50 using quick sort: that is apply partition algorithm recursively. You pick first element of the list as the pivot element and continue until each set generated by partition algorithm is singleton. At each stage show pivot elements in circle or box and show the result of partition in the next line. Show sets subject to partition between parenthesis. Singleton need not be shown between parenthesis. \( \text{[10 pts]} \)

**Solution:** Notation: \( (..) \) denotes elements to be sorted. \( 21 \) represent the pivot element, and \( 21 \) represents an element whose position is fixed within the final sorted list.

\[
\begin{array}{c}
(21) 10 \ 30 \ 5 \ 25 \ 13 \ 35 \ 8 \ 11 \ 28 \ 20 \ 40 \ 50 \\
(20) 10 \ 5 \ 13 \ 8 \ 11 \ 21 \ (25) 30 \ 28 \ 35 \ 40 \ 50 \\
(11) 10 \ 5 \ 13 \ 8 \ 20 \ 21 \ 25 \ (30) 28 \ 35 \ 40 \ 50 \\
(5) \ (8) 10 \ 5 \ 13 \ 20 \ 21 \ 25 \ 28 \ 30 \ (35) 40 \ 50 \\
5 \ 8 \ 10 \ 11 \ 13 \ 20 \ 21 \ 25 \ 28 \ 30 \ 35 \ (40) 50 \\
5 \ 8 \ 10 \ 11 \ 13 \ 20 \ 21 \ 25 \ 28 \ 30 \ 35 \ 40 \ 50
\end{array}
\]
6. Solve the following recursion via Master Theorem, if possible. Show your calculations and justify your results.  

Let \( \alpha = \log_b a \) and \( f(n) = n^\beta g(n) \) where \( g(n) \) does not contain \( n^x \) for any \( x \). 

Master theorem applies only to \( g(n) = \log n^k, k \geq 0 \)

- \( T(n) = 4T(n/2) + \Theta(n^2) \)
  \( \alpha = 2 = \beta \), and \( k = 0 \). Then \( T(n) = O(n^2 \log n) \)

- \( T(n) = 5T(n/2) + \Theta(n^{3/2}) \)
  \( \alpha = \log_2 5 \geq 2 > 3/2 = \beta \) \( \rightarrow T(n) = O(n^\alpha) = O(n^{\log_2 5}) \)

- \( T(n) = T(n/4) + \Theta(\log^2 n) \)
  \( \alpha = \log_4 1 = 0 = \beta \) and \( k = 2 \). Then \( T(n) = O(f(n) \log n) = O(\log^3 n) \)

- \( T(n) = 4T(n/4) + \Theta(n \log n) \)
  \( \alpha = \beta = 1, k = 1 \). Thus \( T(n) = O(f(n) \log n) = O(n \log^2 n) \)

- \( T(n) = 2T(n/3) + \Theta(n \log^2 n) \)
  \( \alpha = \log_3 2 < \beta = 1 \) and \( k = 1 \). Master Theorem does not apply.
Hashing- Double Hashing

Let \( h_1(k) = k \mod 17 \) and \( h'_2(k) = k \mod 11 \) and \( h_2(k) = h'_2(k) \) if \( h'_2(k) > 0 \), and \( h_2(k) = h'_2(k) + 1 \mod 11 \) otherwise. Let \( h(k, i) = (h_1(k) + ih_2(k)) \mod 17 \). Using the sequence \( h(k, i) \ i=0, 1, 2, ... \) place following in a Hash table: 21, 67, 31, 37, 54, 6, 23, 40, 7, 33, 50

<table>
<thead>
<tr>
<th>( k )</th>
<th>21</th>
<th>67</th>
<th>31</th>
<th>37</th>
<th>54</th>
<th>6</th>
<th>23</th>
<th>40</th>
<th>7</th>
<th>33</th>
<th>50</th>
</tr>
</thead>
<tbody>
<tr>
<td>( h_1(k) )</td>
<td>4</td>
<td>16</td>
<td>14</td>
<td>3</td>
<td>3</td>
<td>6</td>
<td>6</td>
<td>7</td>
<td>16</td>
<td>16</td>
<td></td>
</tr>
<tr>
<td>( h_2(k) )</td>
<td>0</td>
<td>10</td>
<td>1</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>( h(k, i) )</td>
<td>4/0</td>
<td>16/0</td>
<td>14/0</td>
<td>3/0</td>
<td>13/1</td>
<td>6/0</td>
<td>7/1</td>
<td>10/3</td>
<td>11/3</td>
<td>0/1</td>
<td>5/1</td>
</tr>
</tbody>
</table>

Solve the same input with \( h_1 \) and linear probing

\( h(k, i) = h_1(k) + i \), for \( i=0, 1, ... \)

| \( k \) | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 |
| 33 | 37 | 21 | 50 | 6 | 23 | 40 | 7 | 54 | 31 | 67 |
8. Build a heap containing the following items: 1 10 30 5 25 35 8 11 15 28 20 40 50 55. Use Heapify!

10 pts

![Heap Diagram](image-url)
9. Given Heap $H_o$, apply successively (3 pts each)

i) insert 300,

ii) insert 175,
ii) apply twice deletemax operation to $H_o$ (3 pts each)
10. Given binary search tree \( T_0 \) apply the following each time to \( T_n \) (5 pts each)

i) insert 14
ii) insert 38

```
   20
  /  \
15   40
 /     \
4      30
   \    /  \
  17   35
   \  / 0
12   38
```

```mermaid
graph LR
A --> C[40]
B --> D[4]
B --> E[17]
C --> F[30]
C --> G[125]
D --> H[12]
D --> I[25]
E --> J[35]
E --> K[0]
F --> L[30]
F --> M[25]
G --> N[38]
G --> O[125]
```

iii) delete 17

```
   20
   /
  /  
25 35
   /
  /
20
```

```
   20
   /
  /  
25 35
   /
  /
20
```

```
   20
   /
  /  
25 35
   /
  /
20
```

```
   20
   /
  /  
25 35
   /
  /
20
```
iv) delete 125