Foundations II: Data Structures and Algorithms

Topic 2:
-- Analyzing algorithms
-- Solving recurrences
Example: *for* Loop (1)

```plaintext
function func(n)
1    x ← 0;
2    for (i = 1; i ++; i ≤ n) do
3        for (j = 1; j ++; j ≤ i) do
4            x ← x + (i - j);
5        end
6    end
7    return (x);
```

- Textbook A.1 for summation formulas

What if *i* (or *j*) starts from 1000?

What if *i* starts from *n/2*?

What if *j* ends with *i*²?

What if *j* ends with √*i*?
Example: \textit{for} Loop (2)

\begin{verbatim}
function func(n) 
1 if (n < 100000) then return (0);
2 x ← 0;
3 x ← 0;
4 for (i = 1; i ++; i ≤ n) do
5     for (j = 1; j ++; j ≤ i) do
6         x ← x + (i - j);
7     end
8 end
9 return (x);
\end{verbatim}

What if we change the first line to $n > 100000$?
Example: *while* Loop

```plaintext
function func(n)
  1 x ← 0;
  2 i ← 7;
  3 while (i ≤ n) do
     4     x ← x + i;
     5     i ← i + 3;
  6 end
  7 return (x);

What if we change the *i* ← *i* + 3 to *i* ← *i* + 100?

What if we change the *i* ← *i* + 3 to *i* ← *i* * 3?
```
Analysis of while Loop

- Stops when $3^k \approx n$, that is, after $k = \Theta(\log_3 n) = \Theta(\log n)$ iterations.

<table>
<thead>
<tr>
<th>iteration</th>
<th>i</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>$3^2$</td>
</tr>
<tr>
<td>3</td>
<td>$3^3$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>j</td>
<td>$3^j$</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>k</td>
<td>n</td>
</tr>
</tbody>
</table>

It does not matter if Changes to $i \leftarrow i \times c$ since $\Theta(\log_c n) = \Theta(\log n)$ for any constant $c$. 
What if we change the $j \leq n$ to $j \leq i$?
Evaluating Summations

- Evaluate $\sum_{i=m}^{n} f(i)$
- No fixed technique
- Be familiar with some basics
  - Textbook Appendix A.1 and A.2
Two Useful Techniques

- Splitting summations
  - E.g, $\sum_{i=1}^{n} i$, $\sum_{i=2}^{n} \log i$

- Using integral:
  - $\sum_{i=1}^{n} f(i) \approx \int_{1}^{n} f(x) \, dx$
  - If $f$ is monotonically increasing,
    $\int_{m-1}^{n} f(x) \, dx \leq \sum_{i=m}^{n} f(i) \leq \int_{m}^{n+1} f(x) \, dx$
  - E.g, $\sum_{i=1}^{n} \sqrt{i}$
More Examples

function func(n)
    1  x ← 0;
    2  for i ← 1 to n do
    3      for j ← 1 to i do
    4          for k ← j to i do
    5              x ← x + (k * i - j);
    6          end
    7      end
    8  end
    9  return (x);
function func(n)
    1  x ← 0;
    2  for i ← 2n to (3n^2 + 5n) do
    3      for j ← 1 to (i^3 + i^2) do
    4          x ← x + (i − j);
    5      end
    6  end
    7 return (x);
More Examples

\[
\text{Func5}(n) \\
1 \quad s \leftarrow 0; \\
2 \quad i = n; \\
3 \quad \textbf{while} \ (i \geq 1) \ \textbf{do} \\
4 \quad \quad s \leftarrow s + s \times i; \\
5 \quad \quad i \leftarrow i/3; \\
6 \quad \textbf{end} \\
7 \quad \textbf{return} \ (s); 
\]
What if we change the $i \leftarrow i \times 2$ to $i \leftarrow i \times 3$?
More Examples

function func(n)
  1  x ← 0;
  2  i ← 1;
  3  while (i ≤ n) do
  4      for j ← 1 to \lfloor n/i \rfloor do
  5          x ← x + (i - j);
  6    end
  7  i ← i + 1;
  8 end

Harmonic series
\[ \sum_{i=1}^{n} \frac{1}{i} = \ln n + o(1) = \Theta(\ln n) \]

What if we change the \( i \leftarrow i + 1 \) to \( i \leftarrow i \times 2 \)?
Geometric Series

- Geometric series
  - \[1 + r + r^2 + \ldots + r^k = \frac{r^{k+1} - 1}{r - 1}\]

- In particular,
  - \(0 < r < 1:\)
    - \[1 + r + r^2 + \ldots + r^k \leq \frac{1}{1-r} \Rightarrow 1 + r + r^2 + \ldots + r^k = \Theta(1)\]
  - \(r > 1: \)
    - \[1 + r + r^2 + \ldots + r^k \leq \frac{r^{k+1}}{r - 1} \Rightarrow 1 + r + r^2 + \ldots + r^k = \Theta(r^k)\]
More Examples (4)

\[
\text{Func2}(n) \\
1 \quad s \gets 0; \\
2 \quad \text{for } i \gets 3 \text{ to } \lceil \sqrt{n} \rceil \text{ do} \\
3 \quad \quad j \gets i^3; \\
4 \quad \quad \text{while } (j \geq i) \text{ do} \\
5 \quad \quad \quad s \gets s + i - j; \\
6 \quad \quad \quad j \gets j - 4; \\
7 \quad \quad \text{end} \\
8 \quad \text{end} \\
9 \quad \text{return } (s);
\]
Recursive Algorithms
Input : Array \( A \) of \( n \) elements.
Result : Permutation of \( A \) such that 

procedure SelectionSort(A[],n)
1    if \((n \leq 1)\) then
2       return;
3    else
4       for \( i \leftarrow 1 \) to \( n - 1 \) do
5           if \((A[i] > A[n])\) then Swap(A[i], A[n]);
6       end
7       SelectionSort(A[],n - 1);
8    end
- **T(n)**: worst case time complexity for any input of size *n*

- Recurrence:
  - \( T(n) = T(n-1) + cn \)

- Solving recurrence:
  - \( T(n) = T(n-1) + cn \)

**Expansion into a series.**

**Usual assumption:**
\( T(n) \) is constant for small *n*, say *n = 1, 2, 3*
More examples (1)

Input : Array A of n integers.

function SelectMax(A[], n)
1   if (n = 1) then
2     return (A[1]);
3   else
4     for i = 1 to [n/2] do
5       A[i] ← max(A[i], A[n − i + 1]);
6     end
7     x ← SelectMax (A[1], [n/2]);
8     return (x);
9   end
function BinarySearchRec(A[,i,j,K)

1 if (j > i) then return (-1);
2 else
3     midp ← [(i + j)/2];
4     if (K = A[midp]) then index ← midp;
5     else if (K < A[midp]) then
6         index ← BinarySearchRec(A,i,midp − 1,K);
7     else /* K > A[midp] */
8         index ← BinarySearchRec(A,midp + 1,j,K);
9     return (index);
10 end
Sorting Revisited (2)

- Can we do better than $\Theta(n^2)$?
- Yes. Many ways.
  - We will talk about Merge-sort
Merge Sort

Divide

Conquer
Conquer Step Merge(B, C)

- Input: Given two sorted arrays B and C
- Output: Merge into a single sorted array

If size of $B$ and $C$ are $s$ and $t$:

Running time: $O(s + t)$
Pseudocode

MergeSort \( (A, r, s) \)

if \( (r \geq s) \) return;

\( m = (r+s) / 2; \)

\( A1 = \text{MergeSort} (A, r, m); \)

\( A2 = \text{MergeSort} (A, m+1, s); \)

Merge \((A1, A2);\)

- Recurrence relation for MergeSort\((A, 1, n)\)
  - \( T(n) = 2T(n/2) + cn \)
Solve Recurrence

- Expansion into a series
- Recursion tree
- Substitution method
Expansion

\[ T(n) = 2 \cdot T(n/2) + n \]
\[ = 2 \cdot (2 \cdot T(n/4) + n/2) + n = 4 \cdot T(n/4) + n + n \]
\[ = 4 \cdot (2 \cdot T(n/8) + n/8)) + n + n \]
\[ \ldots \]
\[ = 2^k \cdot T(n/2^k) + n + \ldots + n \]

Let \( \frac{n}{2^k} = 1 \Rightarrow k = \log_2 n \). Then:

\[ T(n) = \Theta(n \log n) \]
Recursion-tree Method

- Solve $T(n) = 2T(n/2) + n$

$T(n) = \ldots$

$\operatorname{height} \text{tree} = n \lg n$

$\Theta(1)$
Another MergeSort

MergeSort \((A, r, s)\)

if \((r \geq s)\) return;

\[ m_1 = r + (s-r)/3; \]
\[ m_2 = r + 2(s-r)/3; \]

\(A_1 = \) MergeSort \((A, r, m_1)\);

\(A_2 = \) MergeSort \((A, m_1 + 1, m_2)\);

\(A_3 = \) MergeSort \((A, m_2 + 1, s)\);

Merge \((A_1, A_2, A_3)\);

- Recurrence relation for MergeSort\((A, 1, n)\)
  - \(T(n) = 3T(n/3) + cn\)

What if we change 3 to \(k\)?
More Examples of Recurrences (1)

- $T(n) = T(n - 1) + c$
  - $T(n) = \Theta(n)$
- $T(n) = T(n - 1) + cn$
  - $T(n) = \Theta(n^2)$
- $T(n) = T(n - 1) + cn^2$
  - $T(n) = \Theta(n^3)$
More Examples (2)

- \( T(n) = T\left(\frac{n}{2}\right) + c \)
  - \( T(n) = \Theta(\lg n) \)
- \( T(n) = T\left(\frac{n}{3}\right) + cn \)
  - \( T(n) = \Theta(n) \)
- \( T(n) = 2T\left(\frac{n}{2}\right) + cn^2 \)
  - \( T(n) = \Theta(n^2) \)
- \( T(n) = 4T\left(\frac{n}{2}\right) + cn \)
  - \( T(n) = \Theta(n^2) \)
- \( T(n) = 4T\left(\frac{n}{2}\right) + cn^2 \)
  - \( T(n) = \Theta(n^2 \lg n) \)

\[ T(n) = T\left(\frac{n}{4}\right) + T\left(\frac{n}{2}\right) + cn \]
More Examples (3)

- \( T(n) = 2T(n - 1) + c \)
  - \( T(n) = \Theta(2^n) \)

- Sometimes, it is hard to get tight bound
  - \( T(n) = T(n - 1) + T(n - 2) + c \)
    - Easy to prove that \( T(n) = O(2^n) \) and \( T(n) = \Omega(\sqrt{2^n}) \)
Another MergeSort

MergeSort( \( A, r, s \) )

if ( \( r \geq s \) ) return;

\[ m = r + \frac{(s-r)}{3}; \]

\( A1 = \text{MergeSort}( A, r, m ); \)

\( A2 = \text{MergeSort}( A, m+1, s ); \)

Merge( \( A1, A2 \) );

- Recurrence relation for MergeSort(\( A, 1, n \))
  - \( T(n) = T(n/3) + T(2n/3) + cn \)
Recursion Tree

- $\log_3 n$ full levels and $\log_3 n$ total levels
- Total cost of each level is at most $cn$
- Upper bound $O(n \log n)$
- Lower bound $\Omega(n \log n)$
- $T(n) = \Theta(n \log n)$
Summary

- No good formula for solving recurrences
  - Practice!
- Expansion and recursion tree are two most intuitive
  - These two are quite similar
- Substitution method more rigorous and powerful
  - But require good intuition and use of induction
- Other methods:
  - Master Theorem (not covered)