Outline

• Performance analysis
• LZW compression
• Heap
• RB-tree
• B-tree
• Union-Find
• Sorting
Big-Oh

• \( f(N) = O(g(N)) \)

• There are positive constants \( c \) and \( n_0 \) such that
  \[
  f(N) \leq c \cdot g(N) \quad \text{when} \quad N \geq n_0
  \]

• The growth rate of \( f(N) \) is less than or equal to the growth rate of \( g(N) \)

• \( g(N) \) is an upper bound on \( f(N) \)

• Example
  - Let \( f(N) = 2N^2 \). Then
    • \( f(N) = O(N^4) \)
    • \( f(N) = O(N^3) \)
    • \( f(N) = O(N^2) \) (best answer, asymptotically tight)
General Rules (algorithm analysis)

- For loops
  - at most the running time of the statements inside the for-loop (including tests) times the number of iterations.

- Nested for loops
  - the running time of the statement multiplied by the product of the sizes of all the for-loops
  - $O(N^2)$

- Consecutive for loops
  - These just add
  - $O(N) + O(N^2) = O(N^2)$

```cpp
for (i=0;i<n;i++)
    for (j=0;j<n;j++)
        k++;

for (i=0;i<n;i++)
    a[i]=0;
for (i=0;i<n;i++)
    for (j=0;j<n;j++)
        a[i] += a[j]+i+j;
```
Exercise 2.1[w]

• Order the following functions by growth rate:
  – $N$, $N^{1.5}$, $N^2$, $N\log N$, $N\log\log N$, $N\log^2 N$, $N\log(N^2)$, $2/N$, $2^N$, 37, $N^2\log N$, $N^3$

• Answer:
  
  2/N, 37, N, NloglogN, NlogN, NlogN^2, Nlog^2N, N^{1.5}, N^2, N^2logN, N^3, 2^N
Exercise 2.7[w]

• Give an analysis of the Big-Oh running time for the following program fragments:

a.
```
sum = 0;
for (i=0; i<n; i++)
  for (j=0; j<n*n; j++)
    sum++;
```

b.
```
sum = 0;
for (i=0; i<n; i++)
  for (j=0; j<i; j++)
    sum++;
```

c.
```
sum = 0;
for (i=0; i<n; i++)
  for (j=0; j<i*i; j++)
    for (k=0; k<j; k++)
      sum++;
```

d.
```
sum = 0;
for (i=1; i<n; i++)
  for (j=1; j<i*i; j++)
    if (j%i == 0 )
      for (k=0; k<j; k++)
        sum++;
```
Exercise 2.27[w]

- The input is an N by N matrix of numbers that is already in memory. Each individual row is increasing from left to right. Each individual column is increasing from top to bottom. How would you decide if a number X is in the matrix? What is its worst-case complexity in Big-Oh
LZW algorithm

• Encoding:
  – Longest Prefix Matching (e.g. matched string X) in the code table
    • Output the code for X
  – Add a new entry for string Xc, c is a char
  – Move pointer to c’s position
  – Repeat

• Example:
  – abbbbaaab
LZW algorithm

Output: string(first CodeWord);
while(there are more CodeWords) {
    if(CurCodeWord is in the Dict)
        output: string(CurCodeWord)
    else
        output: PreviousOutput + FC(PreviousOutput);
    insert in the Dictionary: PreviousOutput + FC(CurrentOutput);
}

Example:
0 2 3 2 4
Heap

• Usually in array representation

• Key (for Max-Heap):
  – For any node v and its parent p, p.key≥v.key

• Insert
  – Put at the end of the array. Bottom up
  – Delete/Extract Max: Replace the root with the last element. Top-down fix
Heap

• Example
  – We have the following elements in array X
    • $X = [1 \ 3 \ 5 \ 7 \ 9 \ 11 \ 13]$ 
  – Transform the array into a max heap in $O(n)$ time. Draw the resulting max heap

```
  13
  9   11
  7   3   1   5
```
Binomial Heap

• Delete the smallest element from the following Binomial Heap
# Binary Heap & Binomial Heap

<table>
<thead>
<tr>
<th>Procedure</th>
<th>Binomial Heap</th>
<th>Binary Heap</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make-Heap</td>
<td>$O(1)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Insert</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Minimum</td>
<td>$O(\log n)$</td>
<td>$O(1)$</td>
</tr>
<tr>
<td>Extract-Min</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Merge</td>
<td>$O(\log n)$</td>
<td>$O(n\log n) \rightarrow O(n)$</td>
</tr>
<tr>
<td>Decrease-Key</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
<tr>
<td>Delete</td>
<td>$O(\log n)$</td>
<td>$O(\log n)$</td>
</tr>
</tbody>
</table>
Exercise

Insert 70, 80

Insert 26
Exercise

Delete 18
B-tree delete exercise

Delete 16

- Borrow from right sibling
- Exchange with smallest of right sub-tree
Union-find

- Used to maintain equivalent classes.
- Union(x,y): Group x and y into one class
  - Initially each element is a class. N different classes in total
- Find(x): Returns the representative of x’s class
  - If x and y belongs to same class, Find(x) == Find(y)
- Binary Tree/Array Representation.
- Two optimizations:
  - For Union: Weighted Union
  - For Find: Path compression
Union by rank

- Two choices for the union operation

Choice 1: Root = e

Choice 2: Root = h
Path compression

- A little extra computation performed during find, that makes the tree shorter
Exercise

• Show the resulting tree after each union-find operations. Use weighted union & path compression

Find(21), Find(18), Find(36), Find(26), Find(23), Union(19,22)
Heap Sort

• Sort the following array
  – 19, 31, 23, 66, 73, 79, 45, 87
  – Show the array after each iteration
Euler Circuit

• Euler circuit
  – Start from a vertex v, travel all edges exactly once and come back to v
  – Euler theorem: If the degree of any vertex is even, the graph exists an EC
  – Barnard’s algorithm
  – Euler(v)
    • For vertex w that is adjacent to v and (v,w) is not marked
      – Mark (v,w)
      – Path = (v,w) + Euler(w) + Path
    • Return Path
Euler Circuit

• Example
Reference

• Shuo Wu’s review

• ucsd cs101

• sjsu cs146
  – www.cs.sjsu.edu/~lee/cs146/24CS146JCMerge.ppt