Plan for Today

1. Recap

2. Growth of Functions
Plan

1 Recap

2 Growth of Functions
A different sorting algorithm

Algorithm 1: InsertionSort (array $A$ of size $n$)

1. for $i = 2$ to $n$ do 
2. \hspace{1cm} key = $A[i]$ 
3. \hspace{1cm} $j = i - 1$ 
4. \hspace{1cm} while $j > 0$ & $A[j] > key$ do 
5. \hspace{2cm} $A[j+1] = A[j]$ 
6. \hspace{2cm} $j = j - 1$ 
7. \hspace{1cm} $A[j+1] = key$ 
8. return $A$

At the beginning of the $i$-th iteration, $A[1], \ldots, A[i-1]$ has been ordered increasingly.

- $O(n^2)$ comparisons; in-place; stable; online
Still another sorting algorithm

Algorithm 2: BubbleSort (array $A$ of size $n$)

1. for $i = 1$ to $n - 1$ do
2.   isdone = true
3.    for $j = n$ downto $i + 1$ do
6.      isdone = false
7.    if isdone then
8.      break
9. return $A$

- $O(n^2)$ comparisons; in-place; stable
Difference between SelectionSort and BubbleSort

- **SelectionSort**: selects the $i$-th smallest and only swaps it with $A[i]$
- **BubbleSort**: selects the $i$-th smallest, swaps it with $A[i]$ and swaps a bunch of neighbors
Finding

Problem

MaxMin: find maximum and minimum element in $A$

---

**Algorithm 3: MaxMin(array $A$ of size $n$)**

1. $max = A[1]$
2. $min = A[1]$
3. **for** $i = 2$ **to** $n$ **do**
   4. **if** $A[i] > max$ **then**
      5. $max = A[i]$
   6. **else if** $A[i] < min$ **then**
      7. $min = A[i]$
4. **return** $max, min$

---

- Still $O(n)$ comparisons, but can be improved!
Reduction revisited

Algorithm 4: Maximum_Sub(array A of size n)
1  $A = \text{Sort}(A)$
2  return $A[n]$

Algorithm 5: MaxMin_Sub(array A of size n)
1  $A = \text{Sort}(A)$
2  return $A[n], A[1]$

- At least $O(n \log n)$ comparisons; suboptimal
- Reduction is powerful but can be overkill

Never “reduce” a problem to something harder!
A lower bound on comparison sorting

- $n!$ possible permutations.
- A binary tree with depth $h$ can have at most $2^h$ leaves
- $2^h \geq n! \approx n^n \iff h \geq n \log n$.

Figure: courtesy to CLRS
Merge sort

Split, solve, merge, and recurse

Algorithm 6: mSort (array $A$, $p$, $r$)

1. if $p < r$ then
2. $q = \lfloor (p + r)/2 \rfloor$ // split at $q$
3. mSort ($A$, $p$, $q$) // solve the left piece
4. mSort ($A$, $q$, $r$) // solve the right piece
5. Merge ($A$, $p$, $q$, $r$) // merge
The merge function

Algorithm 7: Merge (array $A$, $p$, $q$, $r$)

1. for $i = 1$ to $q - p + 1$ do
2.   $L[i] = A[p + i - 1]$
3. for $j = 1$ to $r - q$ do
5. $i = 1$, $j = 1$
6. for $k = p$ to $r$ do
7.   if $j > r - q$ then  // left part is empty
8.     $A[k] = L[i]$, $i++$
9.   else if $i <= q - p + 1$ && $L[i] \leq R[j]$ then
10.    $A[k] = L[i]$, $i++$  // pick the smaller one
11. else
Analysis of mSort

- Merge costs $O(r - p)$
- with $p = 1$, $r = n$

\[
T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor \right) + T\left(\left\lceil \frac{n}{2} \right\rceil \right) + \Theta(n)
\]

- Claim: $T(n) = \Theta(n \log n)$

This matches our lower bound, hence we have found an optimal algorithm!
Yet another sorting algorithm

Algorithm 8: CountingSort (array A of size n)

1 for \( i = 1 \) to \( n \) do
2 \hspace{1em} count[A[i]]++
3 \hspace{1em} cumsum = 1
4 for \( j = 1 \) to \( k \) do
5 \hspace{2em} c = count[j]
6 \hspace{2em} count[j] = cumsum // starting index of \( j \)
7 \hspace{2em} cumsum += c
8 for \( i = 1 \) to \( n \) do
9 \hspace{2em} S[count[A[i]]] = A[i]
10 \hspace{2em} count[A[i]]++
11 return S

\( O(n + k) \) complexity. No comparison; not in-place; stable.
Plan

1 Recap

2 Growth of Functions
Asymptotic notation

Θ-notation. 
\[ \Theta(g(n)) = \{ f(n) : \exists c_1, c_2 > 0, N > 0, \text{ s.t. } n \geq N \implies 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \} \]

Ω-notation. 
\[ \Omega(g(n)) = \{ f(n) : \exists c > 0, N > 0, \text{ s.t. } n \geq N \implies 0 \leq c g(n) \leq f(n) \} \]

O-notation. 
\[ O(g(n)) = \{ f(n) : \exists c > 0, N > 0, \text{ s.t. } n \geq N \implies 0 \leq f(n) \leq c g(n) \} \]

ω-notation. 
\[ \omega(g(n)) = \{ f(n) : \forall c > 0, \exists N > 0, \text{ s.t. } n \geq N \implies 0 \leq c g(n) < f(n) \} \]

o-notation. 
\[ o(g(n)) = \{ f(n) : \forall c > 0, \exists N > 0, \text{ s.t. } n \geq N \implies 0 \leq f(n) < c g(n) \} \]
Meaning

Interested in the rate of increase of \( f(n) \), usually the time complexity of an algorithm, as \( n \) increases.

- \( f(n) \in \Theta(g(n)) \): \( f(n) \) increases similarly as \( g(n) \)
- \( f(n) \in O(g(n)) \): \( f(n) \) increases slower than or equal as \( g(n) \)
- \( f(n) \in \Omega(g(n)) \): \( f(n) \) increases faster than or equal as \( g(n) \)
- \( f(n) \in o(g(n)) \): \( f(n) \) increases slower than \( g(n) \)
- \( f(n) \in \omega(g(n)) \): \( f(n) \) increases faster than \( g(n) \)

asymptotically, i.e., for \( n \) sufficiently large
How to use asymptotic notations?

- Time complexity $\Theta(f(n))$
- Time complexity $O(f(n))$
- Time complexity $\Omega(f(n))$
- Worst-case time complexity $\Theta(f(n))$
- Worse-case time complexity $O(f(n))$
- Worse-case time complexity $\Omega(f(n))$
Relationships and properties

\[ f(n) \in \Theta(g(n)) \iff g(n) \in \Theta(f(n)) \]
\[ f(n) \in O(g(n)) \iff g(n) \in \Omega(f(n)) \]
\[ f(n) \in o(g(n)) \iff g(n) \in \omega(f(n)) \]
\[ f(n) \in \Theta(g(n)) \iff f(n) \in O(g(n)) \land f(n) \in \Omega(g(n)) \]
\[ f(n) \in o(g(n)) \implies f(n) \in O(g(n)) \]
\[ f(n) \in \omega(g(n)) \implies f(n) \in \Omega(g(n)) \]

\[
\begin{align*}
    f(n) &\in \Theta(f(n)) \\
    f(n) &\leq g(n) \implies O(f(n)) \subseteq O(g(n)) \\
    f(n) &\geq g(n) \implies \Omega(f(n)) \subseteq \Omega(g(n)) \\
    f(n) \in \Theta(g(n)), g(n) \in \Theta(h(n)) &\implies f(n) \in \Theta(h(n))
\end{align*}
\]
Example

\[ f(n) = an^2 + bn + d, \quad a > 0 \]

Claim:
1. \( f(n) = O(n^2) \)
2. \( f(n) = \Omega(n^2) \)
3. \( f(n) = \Theta(n^2) \)
4. \( f(n) \neq \omega(n^{2.000001}) \)
5. \( f(n) \neq o(n^{\sqrt{2}}) \)

Claim: If \( f(n) \) is a polynomial (with positive leading coefficient), then

\[ f(n) = \Theta(n^{\deg f}). \]
The limit definition

Suppose $f(n), g(n) > 0$ for all $n \geq N$, and

$$L = \lim_{n \to \infty} \frac{f(n)}{g(n)}.$$

Then

$$f(n) = \begin{cases} 
\Theta(g(n)), & \text{if } 0 < L < \infty \\
\Omega(g(n)), & \text{if } 0 < L \leq \infty \\
O(g(n)), & \text{if } 0 \leq L < \infty \\
\omega(g(n)), & \text{if } L = \infty \\
o(g(n)), & \text{if } L = 0
\end{cases}$$

Note. Two functions need not be comparable:

$$f(2n + 1) = n, f(2n) = n^3, g(n) = n^2.$$
Algebraic rules

\[ \Theta(C \cdot f(n)) = \Theta(f(n)) \]

\[ \Theta(f(n) + g(n)) = \Theta(\max\{f(n), g(n)\}) \]

\[ \Theta \left( \sum_{k} f(k) \right) = \sum_{k} \Theta(f(k)) \]
Sequences

\[ \Theta \left( \sum_{k=1}^{n} k^d \right) = \sum_{k=0}^{n} \Theta(k^d) = \Theta(n^{d+1}) \]

\[ n^{-1} \sum_{i=0}^{n-1} r^i = \begin{cases} 
\Theta(r^n), & \text{if } r > 1 \\
\Theta(n), & \text{if } r = 1 \\
\Theta(1), & \text{if } r < 1 
\end{cases} \]

\[ n^{-1} \sum_{i=0}^{n-1} i \cdot r^i = \frac{nr^n}{r - 1} + \frac{r(1 - r^n)}{(1 - r)^2}, \quad r \neq 1 \]

\[ H_n := \sum_{i=1}^{n} \frac{1}{i}, \quad \lim_{n \to \infty} (H_n - \ln n) \approx 0.57721 \]
Miscellaneous Formulae

Exponential: \((a^m)^n = a^{mn}, \ a^ma^n = a^{m+n}, \ e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!}\).

Logarithm: \(x = \log_b a \iff a = b^x\).

\[
\log_b x y = \log_b (xy) = \log_b x + \log_b y \\
\log_b x^y = y \log_b x, \ \log_b 1/x = -\log_b x \\
\log_b b = 1, \ \log_b 1 = 0 \\
\log_b a = \frac{\log_c a}{\log_c b}, \ \log_b a = \frac{1}{\log_a b}
\]

\[
a^{\log_b c} = c^{\log_b a}, \ a^x = e^{x \ln a} \\
\log^k a := (\log a)^k, \ \log \log k := \log(\log(k))
\]

All bases are larger than 1; no need to specify.

Factorial: \(n! := 1 \cdot 2 \cdot 3 \cdots n \approx \sqrt{2\pi n} \left(\frac{n}{e}\right)^n\).