Algorithm Analysis (II)

Jordi Cortadella and Jordi Petit
Department of Computer Science
Examples

• Selection sort

• Insertion sort

• The Maximum Subsequence Sum Problem

• Convex Hull
Selection sort uses this invariant:

- This is sorted and contains the i-1 smallest elements.
- This may not be sorted but all elements here are larger than or equal to the elements in the sorted part.
Selection Sort

```python
def selection_sort(v: list[Any]) -> None:
    
    
    for i in range(len(v)-1):
        k = i
        for j in range(i+1, len(v)):
            if v[j] < v[k]:
                k = j;
        v[k], v[i] = v[i], v[k]
```

Algorithm Analysis © Dept. CS, UPC

Observation: notice that \( T(n) \in \Omega(n^2) \), also. Therefore, \( T(n) \in \Theta(n^2) \).
Insertion Sort

• Let us use inductive reasoning:
  – If we know how to sort arrays of size n-1,
  – do we know how to sort arrays of size n?
**Insertion Sort**

```python
def insertion_sort(v: list[Any]) -> None:
    """Sorts v in ascending order""
    for i in range(1, len(v)):  # n-1 times
        x = v[i]
        j = i
        while j > 0 and v[j - 1] > x:  # 0..i times
            v[j] = v[j - 1]
            j -= 1
        v[j] = x
```

Algorithm Analysis

\[
T(n) = \Omega(n)
\]

\[
T(n) = O(n^2)
\]

\[
T_{\text{worst}}(n) = \sum_{i=1}^{n-1} i \cdot O(1) = O(n^2) \quad \Rightarrow \text{sorted in reverse order}
\]

\[
T_{\text{best}}(n) = \sum_{i=1}^{n-1} O(1) = O(n) \quad \Rightarrow \text{already sorted}
\]
The Maximum Subsequence Sum Problem

• Given (possibly negative) integers $A_1, A_2, \ldots, A_n$, find the maximum value of $\sum_{k=i}^{j} A_k$. (the max subsequence sum is 0 if all integers are negative).

• Example:
  – Input: -2, 11, -4, 13, -5, -2
  – Answer: 20 (subsequence 11, -4, 13)

The Maximum Subsequence Sum Problem

def max_sub_sum(a: list[int]) -> int:
    """Returns the sum of the maximum subsequence of a""
    n = len(a)
    max_sum = 0
    # try all possible subsequences
    for i in range(n):
        for j in range(i, n):
            this_sum = 0
            for k in range(i, j+1):
                this_sum += a[k]
                max_sum = max(max_sum, this_sum)
    return max_sum

Algorithm Analysis

\[ T(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^{j} 1 \]
The Maximum Subsequence Sum Problem

\[ T(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^{j} 1 \]

\[ = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1) \]

\[ = \sum_{i=0}^{n-1} \frac{(n - i + 1)(n - i)}{2} = \ldots \]

\[ = \frac{n^3 + 3n^2 + 2n}{6} = \Theta(n^3) \]
def max_sub_sum(a: list[int]) -> int:
    '''Returns the sum of the maximum subsequence of a'''
    n = len(a)
    max_sum = 0
    # try all possible subsequences
    for i in range(n):
        this_sum = 0
        for j in range(i, n):
            this_sum += a[j]  # reuse computation
            max_sum = max(max_sum, this_sum)
    return max_sum

Algorithm Analysis

\[
T(n) = \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} 1 = \Theta(n^2)
\]
Max Subsequence Sum: Divide&Conquer

<table>
<thead>
<tr>
<th>First half</th>
<th>Second half</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>-1</td>
</tr>
<tr>
<td>-3</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>-2</td>
<td>-2</td>
</tr>
</tbody>
</table>

The max sum can be in one of three places:
- 1\textsuperscript{st} half
- 2\textsuperscript{nd} half
- Spanning both halves and crossing the middle

In the 3\textsuperscript{rd} case, two max subsequences must be found starting from the center of the vector (one to the left and the other to the right)
def max_sub_sum_rec(a: list[int], left: int, right: int) -> int:
    """Returns the sum of the maximum subsequence of a[left:right+1]"""
    if left == right:  # base case
        return max(a[left], 0)

    # Recursive cases: left and right halves
    center = (left + right)/2
    max_left = max_sub_sum_rec(a, left, center)
    max_right = max_sub_sum_rec(a, center+1, right)

    # Subsequence in a[center+1:right+1]
    max_rcenter, right_sum = 0, 0
    for i in range(center+1, right+1):
        right_sum += a[i]
        max_rcenter = max(max_rcenter, right_sum)

    # Subsequence in a[left:center+1]
    max_lcenter, left_sum = 0, 0
    for i in range(center, left-1, -1):
        left_sum += a[i]
        max_lcenter = max(max_lcenter, left_sum)

    return max(max_left, max_right, max_lcenter + max_rcenter)
Max Subsequence Sum: Divide&Conquer

\[ T(1) = 1 \]
\[ T(n) = 2T(n/2) + \Theta(n) \]

We will see how to solve this equation formally in the next lesson (Master Theorem). Informally:

\[ T(n) = 2T(n/2) + n = 2(2T(n/4) + n/2) + n \]
\[ = 4T(n/4) + n + n = 8T(n/8) + n + n + n = \cdots \]
\[ = 2^kT(n/2^k) + n + n + \cdots + n \]

when \( n = 2^k \), we have that \( k = \log_2 n \), hence

\[ T(n) = 2^kT(1) + kn = n + n \log_2 n = \Theta(n \log n) \]

But, can we still do it faster?
The Maximum Subsequence Sum Problem

• Observations:
  – If $a[i]$ is negative, it cannot be the start of the optimal subsequence.
  – Any negative subsequence cannot be the prefix of the optimal subsequence.

• Let us consider the inner loop of the $O(n^2)$ algorithm and assume that all prefixes of $a[i..j-1]$ are positive and $a[i..j]$ is negative:

  - If $p$ is an index between $i+1$ and $j$, then any subsequence from $a[p]$ is not larger than any subsequence from $a[i]$ and including $a[p-1]$.
  - If $a[j]$ makes the current subsequence negative, we can advance $i$ to $j+1$. 
The Maximum Subsequence Sum Problem

```cpp
int maxSubSum(const vector<int>& a) {
    int maxSum = 0, thisSum = 0;
    for (int i = 0; i < a.size(); ++i) {
        int thisSum += a[i];
        if (thisSum > maxSum) maxSum = thisSum;
        else if (thisSum < 0) thisSum = 0;
    }
    return maxSum;
}
```

Algorithm Analysis

$T(n) = O(n)$

<table>
<thead>
<tr>
<th>a:</th>
<th>4</th>
<th>-3</th>
<th>5</th>
<th>-4</th>
<th>-3</th>
<th>-1</th>
<th>5</th>
<th>-2</th>
<th>6</th>
<th>-3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>thisSum:</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>maxSum:</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
The Maximum Subsequence Sum Problem

```python
def max_sub_sum(a: list[int]) -> int:
    """Returns the sum of the maximum subsequence of a""
    max_sum, this_sum = 0, 0
    for x in a:
        this_sum += x
        max_sum = max(max_sum, this_sum)
        this_sum = max(this_sum, 0)
    return max_sum
```

Algorithm Analysis

\[ T(n) = \Theta(n) \]

<table>
<thead>
<tr>
<th>a:</th>
<th>4</th>
<th>-3</th>
<th>5</th>
<th>-4</th>
<th>-3</th>
<th>-1</th>
<th>5</th>
<th>-2</th>
<th>6</th>
<th>-3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>this_sum:</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>3</td>
<td>9</td>
<td>6</td>
<td>8</td>
</tr>
<tr>
<td>max_sum:</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>9</td>
</tr>
</tbody>
</table>
A polygon can be represented by a sequence of vertices.

Two consecutive vertices represent an edge of the polygon.

The last edge is represented by the first and last vertices of the sequence.

Vertices: (1,3) (4,1) (7,3) (5,4) (6,7) (2,6)

Edges: (1,3)-(4,1)-(7,3)-(5,4)-(6,7)-(2,6)-(1,3)

// A polygon (an ordered set of vertices)
using Polygon = vector<Point>;
Create a polygon from a set of points

Given a set of $n$ points in the plane, connect them in a simple closed path.
Simple polygon

- **Input:** \( p_1, p_2, \ldots, p_n \) (points in the plane).
- **Output:** \( P \) (a polygon whose vertices are \( p_1, p_2, \ldots, p_n \) in some order).

- Select a point \( z \) with the largest \( x \) coordinate (and smallest \( y \) in case of a tie in the \( x \) coordinate). Assume \( z = p_1 \).
- For each \( p_i \in \{ p_2, \ldots, p_n \} \), calculate the angle \( \alpha_i \) between the lines \( z - p_i \) and the \( x \) axis.
- Sort the points \( \{ p_2, \ldots, p_n \} \) according to their angles. In case of a tie, use distance to \( z \).
Simple polygon
Simple polygon

Implementation details:

- There is no need to calculate angles (requires arctan). It is enough to calculate slopes ($\Delta y/\Delta x$).
- There is not need to calculate distances. It is enough to calculate the square of distances (no sqrt required).

**Complexity:** $O(n \log n)$. The runtime is dominated by the sorting algorithm.
Compute the convex hull of $n$ given points in the plane.
Clockwise and counter-clockwise

How to calculate whether three consecutive vertices are in a clockwise or counter-clockwise turn.

**counter-clockwise**

\(p_3\) at the left of \(\overrightarrow{p_1p_2}\)

\[\alpha < \beta\]

**clockwise**

\(p_3\) at the right of \(\overrightarrow{p_1p_2}\)

\[\alpha > \beta\]

# Returns true if \(p_3\) is at the left of \(\overrightarrow{p_1p_2}\)

```python
def leftof(p1, p2, p3):
    return (p2.x - p1.x) * (p3.y - p1.y) > (p2.y - p1.y) * (p3.x - p1.x)
```
Convex hull: gift wrapping algorithm

Convex hull: gift wrapping algorithm

- **Input:** \( p_1, p_2, \ldots, p_n \) (points in the plane).
- **Output:** \( P \) (the convex hull of \( p_1, p_2, \ldots, p_n \)).

- **Initial points:**
  \( p_0 \) with the smallest \( x \) coordinate.

- **Iteration:** Assume that a partial path with \( k \) points has been built (\( p_k \) is the last point). Pick some arbitrary \( p_{k+1} \neq p_k \). Visit the remaining points. If some point \( q \) is at the left of \( p_k p_{k+1} \) redefine \( p_{k+1} = q \).

- Stop when \( P \) is complete (back to point \( p_0 \)).

**Complexity:** At each iteration, we calculate \( n \) angles. \( T(n) = O(hn) \), where \( h \) is the number of points in the convex hull. In the worst case, \( T(n) = O(n^2) \).
Polygon convexHull(const vector<Point>& points) {
    int n = points.size();
    Polygon hull;

    // Pick the leftmost point
    int left = 0;
    for (int i = 1; i < n; i++)
        if (points[i].x < points[left].x) left = i;

    int p = left;
    do {
        hull.push_back(points[p]);  // Add point to the convex hull

        int q = (p + 1) % n;  // Pick a point different from p
        for (int i = 0; i < n; i++)
            if (leftof(points[p], points[q], points[i])) q = i;

        p = q;  // Leftmost point for the convex hull
    } while (p != left);  // While not closing polygon

    return hull;
}
Convex hull: Graham Scan

https://en.wikipedia.org/wiki/Graham_scan
Convex hull: Graham scan

Input: \( p_1, p_2, \ldots, p_n \) (points in the plane).
Output: \( q_1, q_2, \ldots, q_m \) (the convex hull).

Initially:
Create a simple polygon \( P \) (complexity \( O(n \log n) \)).
Assume the order of the points is \( p_1, p_2, \ldots, p_n \).

// \( Q = (q_1, q_2, \ldots) \) is a vector where the points
// of the convex hull will be stored.
\[
q_1 = p_1; \quad q_2 = p_2; \quad q_3 = p_3; \quad m = 3;
\]
for \( k = 4 \) to \( n \):

\[
\text{while leftof}(q_{m-1}, q_m, p_k): \quad m = m - 1;
\]
\[
m = m + 1;
\]
\[
q_m = p_k;
\]

**Observation:** each point \( p_k \) can be included in \( Q \) and deleted at most once.
The main loop of Graham scan has linear cost.

**Complexity:** dominated by the creation of the simple polygon \( \rightarrow O(n \log n) \).
EXERCISES
Summations

Prove the following equalities:

\[ \sum_{i=1}^{n} i = \frac{n(n + 1)}{2} \]

\[ \sum_{i=1}^{n} i^2 = \frac{n(n + 1)(2n + 1)}{6} \]

\[ \sum_{i=0}^{n} 2^i = 2^{n+1} - 1 \]
For loops: analyze the cost of each code

Calculate the value of variable \( s \) and the end the code

```python
# Code 1
s = 0
for i in range(n):
    s += 1

# Code 2
s = 0
for i in range(0, n, 2):
    s += 1

# Code 3
s = 0
for i in range(n):
    s += 1
for j in range(n):
    s += 1

# Code 4
s = 0
for i in range(n):
    for j in range(n):
        s += 1

# Code 5
s = 0
for i in range(n):
    for j in range(i):
        s += 1

# Code 6
s = 0
for i in range(n):
    for j in range(i, n):
        s += 1
```
For loops: analyze the cost of each code

# Code 7
s = 0
for i in range(n):
    for j in range(n):
        for k in range(n):
            s += 1

# Code 8
s = 0
for i in range(n):
    for j in range(i):
        for k in range(j):
            s += 1

# Code 9
s = 0
i = 1
while i <= n:
    s += 1
    i *= 2

# Code 10
s = 0
for i in range(n):
    j = 1
    while j <= n:
        s += 1
        j *= 2

# Code 11
s = 0
for i in range(n):
    for j in range(i*i):
        for k in range(n):
            s += 1

# Code 12
s = 0
for i in range(n):
    for j in range(i*i):
        if j%i == 0:
            for k in range(n):
                s += 1
The following statements refer to the *insertion sort* algorithm and the X’s hide an occurrence of $O$, $\Omega$ or $\Theta$. For each statement, find which options for $X \in \{O, \Omega, \Theta\}$ make the statement true or false. Justify your answers.

1. The worst case is $X(n^2)$
2. The worst case is $X(n)$
3. The best case is $X(n^2)$
4. The best case is $X(n)$
5. For every probability distribution, the average case is $X(n^2)$
6. For every probability distribution, the average case is $X(n)$
7. For some probability distribution, the average case is $X(n \log n)$
The following algorithms try to determine whether $n \geq 0$ is prime. Find which ones are correct and analyze their cost as a function of $n$.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>Code</th>
</tr>
</thead>
</table>
| is_prime1 | def is_prime1(n: int) -> bool:  
  if n <= 1:  
    return False  
  for i in range(2,n):  
    if n%i == 0:  
      return False  
  return True |
| is_prime2 | def is_prime2(n: int) -> bool:  
  if n <= 1:  
    return False  
  for i in range(2, int(math.sqrt(n))):  
    if n%i == 0:  
      return False  
  return True |
| is_prime3 | def is_prime3(n: int) -> bool:  
  if n <= 1:  
    return False  
  for i in range(2, round(math.sqrt(n))):  
    if n%i == 0:  
      return False  
  return True |
| is_prime4 | def is_prime4(n: int) -> bool:  
  if n <= 1:  
    return False  
  for i in range(2, int(math.sqrt(n))+1):  
    if n%i == 0:  
      return False  
  return True |
| is_prime5 | def is_prime5(n: int) -> bool:  
  if n <= 1:  
    return False  
  if n == 2:  
    return True  
  if n%2 == 0:  
    return False  
  for i in range(3, int(math.sqrt(n))+1, 2):  
    if (n%i == 0):  
      return False  
  return True |
The Sieve of Eratosthenes

The following program is a version of the Sieve of Eratosthenes. Analyze its complexity.

```python
def primes(n: int) -> list[bool]:
    p: list[bool] = [True]*(n+1)
    p[0] = p[1] = False
    for i in range(2, int(math.sqrt(n))+1):
        if p[i]:
            for j in range(i*i, n+1, i):
                p[j] = False
    return p
```

You can use the following equality, where $p \leq x$ refers to all primes $p \leq x$:

$$\sum_{p \leq x} \frac{1}{p} = \log \log x + O(1)$$
The Cell Phone Dropping Problem

- You work for a cell phone company which has just invented a new cell phone protector and wants to advertise that it can be dropped from the $f^{th}$ floor without breaking.

- If you are given 1 or 2 phones and an $n$ story building, propose an algorithm that minimizes the worst-case number of trial drops to know the highest floor it won’t break.

- Assumption: a broken cell phone cannot be used for further trials.

- How about if you have $p$ cell phones?

(Source: Wood & Yasskin, Texas A&M University)