ECS 122A: Algorithm Design and Analysis

Week 1 Discussion

Ji Wang

Spring 2020

Outline

- Mathematical Induction
- Insertion Sort and Time Analysis
- Merge Sort and Recurrence Relation
- Asymptotic Notation

Mathematical Induction: Approach

When applicable: Prove that a property $P(n)$ holds for every natural number n.

How it works:

1. Show that a property $P(n)$ holds for the basis case, usually when $n = 0$ or 1.
2. Assume that $P(n)$ is true for $n = k$ where k is greater than the basis case.
3. Prove that $P(n)$ is true for $n = k + 1$. In this step we will use the assumption above.
4. Then, by the principle of induction, we can conclude that $P(n)$ is true for every natural number that is greater than or equal to the basis case.

Mathematical Induction: Example

Problem Statement: Prove the following statement $P(n)$ is true

$$1^3 + 2^3 + \cdots + n^3 = (\frac{n(n+1)}{2})^2$$

Build a skeleton:

- Basis Step:
- Inductive Hypothesis:
- Inductive Step:
- Thus, by the principle of induction, we can conclude that the statement above is true for every natural number n.

Mathematical Induction: Example

$$1^3 + 2^3 + \cdots + n^3 = (\frac{n(n+1)}{2})^2$$

- Basis Step:
  Verify $P(n)$ is true when $n = 1$. $1^3 = (\frac{1\cdot(1+1)}{2})^2$.

Mathematical Induction: Example

$$1^3 + 2^3 + \cdots + n^3 = (\frac{n(n+1)}{2})^2$$

- Basis Step:
  Verify $P(n)$ is true when $n = 1$. $1^3 = (\frac{1 \cdot (1+1)}{2})^2$.
- Inductive Hypothesis:
  Assume $P(n)$ is true when $n = k$. Then, we have
  $1^3 + 2^3 + \cdots + k^3 = (\frac{k(k+1)}{2})^2$.

Mathematical Induction: Example

$$1^3 + 2^3 + \cdots + n^3 = (\frac{n(n+1)}{2})^2$$

- Inductive Step: We need to show $P(n)$ is true when $n = k + 1$.

$$\begin{aligned}
1^3 + 2^3 + \cdots + (k+1)^3 &= 1^3 + 2^3 + \cdots + k^3 + (k+1)^3 \\
&= (\frac{k(k+1)}{2})^2 + (k+1)^3 \\
&= \frac{k^2(k+1)^2}{4} + \frac{4(k+1)^3}{4} \\
&= \frac{(k+1)^2(k^2 + 4(k+1))}{4} \\
&= \frac{(k+1)^2(k+2)^2}{4} \\
&= (\frac{(k+1)((k+1)+1)}{2})^2
\end{aligned}$$

Insertion Sort: Example

Problem Statement: Given a list of numbers, sort them in non-decreasing order.

Algorithm: Split the list into sorted part and unsorted part. Insert the head of unsorted one into the appropriate position in the sorted one. Repeat until no unsorted part remaining.

Example: 8 3 2 7 4 1 6

$\boxed{8}$ 3 2 7 4 1 6

$\boxed{8\ 3}$ 2 7 4 1 6

$\boxed{3\ 8}$ 2 7 4 1 6

$\boxed{3\ 8\ 2}$ 7 4 1 6

$\boxed{2\ 3\ 8}$ 7 4 1 6

$\boxed{2\ 3\ 8\ 7}$ 4 1 6

$\boxed{2\ 3\ 7\ 8}$ 4 1 6

$\boxed{2\ 3\ 7\ 8\ 4}$ 1 6

$\boxed{2\ 3\ 4\ 7\ 8}$ 1 6

$\boxed{2\ 3\ 4\ 7\ 8\ 1}$ 6

$\boxed{1\ 2\ 3\ 4\ 7\ 8}$ 6

$\boxed{1\ 2\ 3\ 4\ 7\ 8\ 6}$

$\boxed{1\ 2\ 3\ 4\ 6\ 7\ 8}$

Insertion Sort: Pseudocode and Time Analysis

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert key into the sorted list A[1 ··· j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

See Section 2.1 on textbook for pseudocode convention.

- Worst case scenario: reversed order, e.g. 5 4 3 2 1.

Insertion Sort: Pseudocode and Time Analysis

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert key into the sorted list A[1 ··· j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

See Section 2.1 on textbook for pseudocode convention.

- Worst case scenario: reversed order, e.g. 5 4 3 2 1.
- Running time in this case: $\sum_{j=2}^{n} \sum_{i=1}^{j-1} = O(n^2)$

Insertion Sort: Pseudocode and Time Analysis

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert key into the sorted list A[1 ··· j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

See Section 2.1 on textbook for pseudocode convention.

- Worst case scenario: reversed order, e.g. 5 4 3 2 1.
- Running time in this case: $\sum_{j=2}^{n} \sum_{i=1}^{j-1} = O(n^2)$
- Best case scenario: already sorted

Insertion Sort: Pseudocode and Time Analysis

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert key into the sorted list A[1 ··· j − 1].
    i = j − 1
    while i > 0 and A[i] > key
        A[i + 1] = A[i]
        i = i − 1
    A[i + 1] = key
```

See Section 2.1 on textbook for pseudocode convention.

- Worst case scenario: reversed order, e.g. 5 4 3 2 1.
- Running time in this case: $\sum_{j=2}^{n} \sum_{i=1}^{j-1} = O(n^2)$
- Best case scenario: already sorted
- Running time in this case: $\sum_{j=2}^{n} = O(n)$

Merge Sort: Divide-and-Conquer Approach

1. **Divide** the problem into a number of subproblems that are smaller instances of the **same** problem.
2. **Conquer** by solving the subproblems **recursively**.
3. **Combine** the solutions to the subproblems to produce the solution to the original problem.

MERGE-SORT(A, p, r)

if $p < r$ // Input list: $A[p \cdots r]$
    $q = \lfloor (p + r)/2 \rfloor$ // Divide
    MERGE-SORT(A, p, q) // Conquer
    MERGE-SORT$(A, q + 1, r)$ // Conquer
    MERGE(A, p, q, r) // Combine

Merge Sort: How to combine

$\boxed{L_1 \;\; L_2 \;\; L_3 \;\; L_4 \;\; \infty}$ $\qquad$ $\boxed{R_1 \;\; R_2 \;\; R_3 \;\; R_4 \;\; \infty}$

$$\boxed{A_1 \;\; A_2 \;\; A_3 \;\; A_4 \;\; A_5 \;\; A_6 \;\; A_7 \;\; A_8}$$

Comparison part of `Merge` function:

```
i = 1
j = 1
for k = 1 to A.length
        if L[i] ≤ R[j]
                A[k] = L[i]
                i = i + 1
        else
                A[k] = R[j]
                j = j + 1
```

For full version, refer to the textbook.

Merge Sort: Recurrence Relation

```
MERGE-SORT(A, p, r)
if p < r                          // Input list: A[p ··· r]
    q = ⌊(p + r)/2⌋               // Divide
    MERGE-SORT(A, p, q)           // Conquer
    MERGE-SORT(A, q + 1, r)       // Conquer
    MERGE(A, p, q, r)             // Combine
```

$$T(n) = 2T(\frac{n}{2}) + n$$

- Where does n come from? # of comparisons in `Merge` function.

Merge Sort: Recurrence Relation

```
MERGE-SORT(A, p, r)
if p < r                          // Input list: A[p ··· r]
    q = ⌊(p + r)/2⌋               // Divide
    MERGE-SORT(A, p, q)           // Conquer
    MERGE-SORT(A, q + 1, r)       // Conquer
    MERGE(A, p, q, r)             // Combine
```

$$T(n) = 2T(\frac{n}{2}) + n$$

- Where does n come from? # of comparisons in `Merge` function.
- Why n? As long as a comparison is done, we assign a value to an element in the merged array.

Asymptotic Notation

By applying Limit Lemma Theorem, we can determine the growth relation between functions.

Frequently used functions	Order of Growth	Example
constant	$O(1)$	$f(n) = 10$
logarithm	$O(\log n)$	$f(n) = 4\log(n+2)$
linear	$O(n)$	$f(n) = 2n + 1$
linearithmic	$O(n \log n)$	$f(n) = 3n\log 4n + 1$
quadratic	$O(n^2)$	$f(n) = 2n^2 + 3n + 1$
cubic	$O(n^3)$	$f(n) = n^3 + 1$
exponential	$O(a^n), a \geq 2$	$f(n) = 3^n$

More on Auxiliary Notes.