**CS445**  
*Introduction to Algorithms*  
Instructor: **Alon Efrat**

Some slides are courtesy of  
Piotr Indyk and Carola Wenk

---

**Polices**

---

**Homeworks**

- Alg: Once homeworks are published (and unless stated otherwise)
  - Read questions
  - If needed, re-watch lectures (Alon and Others) online,
  - Thinks really hard. Discover what does *not* work and why
  - Meet your peers (no more than two of them) and and discuss.
  - Write Solutions yourself.
- Diverging from this algorithm might improve your hw grade but is likely to impact your exams grades (not to mention ethical issues, honor code etc).
- Homework's rules.
  - Collaborations ++. Brainstorming in small groups (*max 4*).
  - Give credit. Specify your contribution to each solution (in %).
  - Sharing text is cheating.

---

**CS445 - Regulation, Bureaucracy**

1. Grading Scheme (midterm vs. final)
2. Textbooks
3. Video recording
4. Web Resources
5. Prerequisites (course is mostly self contained, but harder if you did not pass cs345.
   I. Post are for clarifications about the questions.
   II. Be careful not to share any hints in your posts Eg. "*Could we use Quicksort for the solution of hw3 Q?*" is a violation of code of conduct and should not be asked *publically.*
   "*Does anybody else feels that...?*
7. If you have any doubts, send a private message.
8. Attendance

---

**Could we use screens in the classroom?**

1. Screens - allowable, but your focus should be centered at the lecture.
2. Keep regular eye contact with me.
3. If you hear ”no screens for you today” it is not an insult.

---

**Class participations**

- Class participations might effect your grade positively.
- Class participation is expressed via
  1. Asking questions in the classroom
  2. Answering lecturer’s questions in the classroom
  3. Office hours
  4. Piazza discussions
Why these particular algorithms??

- In this course, we will discuss problems, and algorithms for solving these problems.
- There are so many algorithms – why focus on the ones in the syllabus?

Why these algorithms (cont.)

- Relevance to many areas:
  - E.g., networking, search engines, …
- Coolness
- Demonstrate important non-trivial ideas that could be used as building blocks to solutions to other problems that are not discussed in the course (more on it next slide)
- Main Paradigms

Main paradigms:
  a) Greedy algorithms
  b) Divide-and-Conquer
  c) Dynamic programming
  d) Randomized incremental
  e) Branch-and-Bound (mostly in AI)
  f) Etc etc.

Next – Some Sorting Algorithms

- Good occasion to go over two types of sorting algorithms (you are suppose to know more from CSc345)
- Practice asymptotic time complexity notations.

The problem of sorting

**Input**: sequence \( \langle a_1, a_2, \ldots, a_n \rangle \) of numbers.

**Output**: permutation \( \langle a'_1, a'_2, \ldots, a'_n \rangle \) such that \( a'_1 \leq a'_2 \leq \cdots \leq a'_n \).

Example:

**Input**: 8 2 4 9 3 6

**Output**: 2 3 4 6 8 9

Insertion sort

**A**: 1 5 7 10 12 18 9 100 200

**Invariants**: the subarray \( A[1..j-1] \) is sorted

- Consider \( A[j] = 9 \). Obviously violating the invariant. (Not in the correct place.)
- Need to make room for 9 somewhere to its left.
- Need to shift all elements right, starting from 10.
**Insertion sort**

**Pseudocode**

```
Insertion-Sort(A, n)  //input: A[1..n]
    for j ← 2 to n  //outer loop
        do key ← A[j]
            i ← j – 1
            while i > 0 and A[i] > key  //inner loop
                do { A[i+1] ← A[i]
                      i ← i – 1}
            A[i+1] ← key
```

Example of insertion sort

```
8 2 4 9 3 6
```

```
2 8 4 9 3 6
```

```
2 4 8 9 3 6
```
Running time

- The running time depends on the input: an already sorted sequence is easier to sort.
- Parameterize the running time by the size of the input \( n \)
- Seek upper bounds on the running time \( T(n) \) for the input size \( n \), because everybody likes a guarantee.

Machine-independent time

*What is insertion sort’s worst-case time?*

- It depends on the speed of our computer:
  - relative speed (on the same machine),
  - absolute speed (on different machines).

**BIG IDEA:**
- Ignore machine-dependent constants.
- Look at growth of \( T(n) \) as \( n \to \infty \).

“Asymptotic Analysis”

**\( \bigO \)-notation**

Let \( T(n) \) denote (say) the running time of the insertion sort algorithm on an input of \( n \) keys. We say that \( T(n) = \bigO(g(n)) \) iff there exists positive constants \( c_1 \) and \( n_0 \) such that

\[
0 \leq T(n) \leq c_1 g(n) \quad \text{for all } n \geq n_0
\]

In this course, in most cases, \( g(n) \) is one of the functions

- \( \log n \)
- \( n \log n \)
- \( n^2 \)

**Engineering:**

- Drop low-order terms; ignore leading constants.
- Example: \( 3n^3 + 90n^2 - 5n + 6046 = \bigO(n^3) \)

**\( \Theta \)-notation**

*Math:

we say that \( T(n) = \Theta(g(n)) \) iff there exist positive constants \( c_1, c_2, \) and \( n_0 \) such that

\[
0 \leq c_1 g(n) \leq T(n) \leq c_2 g(n) \quad \text{for all } n \geq n_0
\]

**Engineering:**

- Drop low-order terms; ignore leading constants.
- Example: \( 3n^3 + 90n^2 - 5n + 6046 = \Theta(n^3) \)

Asymptotic performance

When \( n \) gets large enough, a \( \Theta(n^2) \) algorithm **always** beats a \( \Theta(n^3) \) algorithm.

- We shouldn’t ignore asymptotically slower algorithms, however.
- Real-world design situations often call for a careful balancing of engineering objectives.
- Asymptotic analysis is a useful tool to help to structure our thinking.
**O-notation - cont**

So if $T(n) = \Theta(n^2)$ then we are also sure that $T(n) = O(n^2)$ and that $T(n) = O(n^{2.5})$ and $T(n) = O(2^n)$.

---

**Insertion sort analysis**

**Worst case:** Input reverse sorted.

$$T(n) = 2c + 3c + 4c + \ldots + c(n-1) = cn(n-1)/2$$

$c$ is a constant (the time to perform a basic operation)

**[arithmetic series]**

<table>
<thead>
<tr>
<th>j</th>
<th>$\Theta(j)$</th>
<th>$\Theta(n^2)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>$x$</td>
<td>$x$</td>
</tr>
<tr>
<td>3</td>
<td>$xx$</td>
<td>$xx$</td>
</tr>
<tr>
<td>4</td>
<td>$xxx$</td>
<td>$xxx$</td>
</tr>
<tr>
<td>n-1</td>
<td>$x\ldots xx$</td>
<td>$x\ldots xx$</td>
</tr>
</tbody>
</table>

Is insertion sort a fast sorting algorithm?

- Moderately so, for small $n$.
- Not at all, for large $n$.

---

**Merge sort**

*(divide-and-conquer algorithm)*

**MERGE-SORT $A[1 \ldots n]$**

1. If $n = 1$, done.
2. Recursively sort $A[1 \ldots \lceil n/2 \rceil]$ and $A[\lceil n/2 \rceil+1 \ldots n]$.
3. “Merge” the 2 sorted lists.

**Key subroutine:** **MERGE**

---

**Merging two sorted arrays**

<table>
<thead>
<tr>
<th>20</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

---

Merging two sorted arrays

<table>
<thead>
<tr>
<th>20</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

---

Merging two sorted arrays

<table>
<thead>
<tr>
<th>20</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
</tbody>
</table>

---

Merging two sorted arrays

<table>
<thead>
<tr>
<th>20</th>
<th>12</th>
</tr>
</thead>
<tbody>
<tr>
<td>13</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>
Merging two sorted arrays

Time = \Theta(n) to merge a total of \( n \) elements (linear time).

Analyzing merge sort

Recurrence for merge sort

\[
T(n) = \begin{cases} 
\Theta(1) \text{ if } n = 1; \\
2T(n/2) + \Theta(n) \text{ if } n > 1. 
\end{cases}
\]

- We shall usually omit stating the base case when \( T(n) = \Theta(1) \) for sufficiently small \( n \), but only when it has no effect on the asymptotic solution to the recurrence.
- CLRS provides several ways to find a good bound on \( T(n) \).
Recursion tree
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

Recursion tree
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

Recursion tree
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

Recursion tree
Solve $T(n) = 2T(n/2) + cn$, where $c > 0$ is constant.

$h = \log_2 n$
Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{array}{c}
T(n) = 2T(n/2) + cn \\
\Theta(1) \rightarrow \ldots \rightarrow \Theta(1)
\end{array}
\]

\( h = \log n \frac{cn}{4} \)

\#leaves = \#leaves = \Theta(n)

Total = \Theta(n \log n)

Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{array}{c}
T(n) = 2T(n/2) + cn \\
\Theta(1) \rightarrow \ldots \rightarrow \Theta(1)
\end{array}
\]

\( h = \log n \frac{cn}{4} \)

\#leaves = \#leaves = \Theta(n)

Total = \Theta(n \log n)

Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{array}{c}
T(n) = 2T(n/2) + cn \\
\Theta(1) \rightarrow \ldots \rightarrow \Theta(1)
\end{array}
\]

\( h = \log n \frac{cn}{4} \)

\#leaves = \#leaves = \Theta(n)

Total = \Theta(n \log n)

Recursion tree

Solve \( T(n) = 2T(n/2) + cn \), where \( c > 0 \) is constant.

\[
\begin{array}{c}
T(n) = 2T(n/2) + cn \\
\Theta(1) \rightarrow \ldots \rightarrow \Theta(1)
\end{array}
\]

\( h = \log n \frac{cn}{4} \)

\#leaves = \#leaves = \Theta(n)

Total = \Theta(n \log n)

Conclusions

- \( \Theta(n \log n) \) grows more slowly than \( \Theta(n^2) \).
- Therefore, merge sort asymptotically beats insertion sort in the worst case.
- In practice, merge sort beats insertion sort for \( n > 30 \) or so.
- Go test it out for yourself!
More examples – iterative method (1)

Recursion formula: \( T(n) = c + T(n-1) \), where \( T(1) = c \). We can solve it using the iteration method:

\[
T(n) = c + T(n-1) = c + c + T(n-2) = 2c + T(n-3) = \ldots = c + c + \ldots + \text{(pick } k < n\text{)} \\
k_c + T(n-k) = \text{(setting } k = n-1\text{)} \\
(n-1)c + T(1) = nc
\]

Example 2

Recursion formula: \( T(n) = cn + T(n-1) \), where \( T(1) = c \). We can solve it using the iteration method:

\[
T(n) = cn + T(n-1) = cn + (cn-1) + T(n-2) = \ldots = cn + \ldots + cn + T(1) = cn(n+1)/2 = \Theta(n^2).
\]

Example 3

Read(n), \( k = 1 \)

while (\( k \times 2 \leq n \)) \( k = 2k \)

We know that each iteration takes \( O(1) \) times. Need to find the number of iterations.

- After the first iteration \( k = 2 \)
- After the 2nd iteration \( k = 4 \)
- After the 3rd iteration \( k = 8 \)
- ...After the \( j \)th iteration \( k = 2^j \)
- Assume \( j \) iterations occurs until the loop exits. After the last one we have that \( k = 2^j < 2n \).

Taking \( \log k \) from both sides, we have that

\[
\log k = \log 2^j < \log 2n \quad \text{or} \quad j \log 2 < \log n \quad \text{or} \quad j \leq \log n + 1 \quad \text{or} \quad j = O(\log n).
\]

\( T(n) = O(\log n) \)

Homework: Prove \( T(n) = O(\log n) \)

Example 4

Read(n), \( k = 1 \)

for \( i = 1 \) to \( n \)

for \( j = i \) to \( k \)

print \( * \)

\( T(n) = cn + n/2 + n/3 + \ldots + n/n = n(1/2 + 1/3 + \ldots + 1/n) = n \ln n \)

Harmonic Sum

Exampels

Recursion formula: \( T(n) = c + T(n-1) \), where \( T(1) = c \). We can solve it using the iteration method:

\[
T(n) = c + T(n-1) = c + c + \ldots + \text{(pick } k < n\text{)} \\
k_c + T(n-k) = \text{(setting } k = n-1\text{)} \\
(n-1)c + T(1) = nc
\]

Exampels
Properties of big-O

- **Claim:** if \( T_1(n) = O(g(n)) \) and \( T_2(n) = O(g(n)) \) then \( T_1(n) + T_2(n) = O(g(n)) \)

- **Example:** \( T_1(n) = O(n^2) \), \( T_2(n) = O(n \log n) \) then \( T_1(n) + T_2(n) = O(n^2) \) \( = O(n \log n) \)

- **Proof:** We know that there are constants \( n_1, n_2, c_1, c_2 \) s.t.
  - for every \( n > n_1 \), \( T_1(n) < c_1 g(n) \)
  - for every \( n > n_2 \), \( T_2(n) < c_2 g(n) \)

  \[ T_1(n) + T_2(n) < c_1 g(n) + c_2 g(n) \leq c' g(n) + c'' g(n) = c (g(n) + g(n)) \]

More properties of big-O

- **Claim:** if \( T(n) = O(g(n)) \) and \( T(n) = O(g(n)) \) then \( T(n) = O(g(n)) \)

- **Example:** \( T(n) = O(n^{a^2}) \), \( T(n) = O(n \log n) \) then \( T(n) = O(n \log n) \)

- Similar properties hold for \( \Theta \), \( \Omega \)

Example 2 (another look)

Read(1), \( i=0 \) to \( i=31415926 \)
for \( i=1 \; i<=n \; i++ \) \{ \( n_i \; n_{i+1} \) \}

Recursion formula: \( T(n) = cn + T(n-1) \), where \( T(1) = c \). We can solve it using the iteration method:

\[
T(n) = cn + T(n-1) = cn + [c(n-1) + T(n-2)] = \]
\[
cn + [c(n-1) + c(n-2) + T(n-3)] = \]
\[
c^n + n+1 + n+2 + n+3 + \ldots + n + \text{pick } k < n \]
\[
= c(n + (n-1) + (n-2) + \ldots + 2 + 1) = T(n-1)
\]

\[
\text{setting } k = n-1 \ldots \]
\[
c^n + n + (n-1) + (n-2) + \ldots + 1 \times T(1) =
\]
\[
c^{n+1} + 1 + 2 + 3 + \ldots + n \times T(1) = cn(n+1)/2 = \mathcal{O}(n^2)
\]

More about \( \Omega(\cdot) \)

Sometimes we would talk about a lower bound on the running time of a specific algorithm.

E.g. The insertion sort might take \( \Omega(n^2) \) for some input.

Sometimes we would talk about a lower bound on the running time of a problem.

E.g.
1. Any algorithm that reads all the input (for any problem) requires \( \Omega(n^2) \) time.
2. Any algorithm that stores all the data requires \( \Omega(n) \) space.
3. Any algorithm that sort \( n \) keys requires \( \Omega(n \log n) \)
   (disclaimer – could be better if we make some assumptions about the keys or the model. Usually:
   - Sorting sort integers takes \( \Omega(n) \) (how?)
   - Sorting floats takes \( \Omega(n \log n) \)

The lower bound trick

On the board
The lower bound trick – Second example
We are about to insert $n$ keys $\{k_1, \ldots, k_n\}$ into an empty AVL tree. How much time would it take?

Upper bound: When the $i+1$ key is inserted, the tree contains $i$ keys, so its height is $O(\log i)$, and an insert operation takes $O(\log i)$ which is also $O(\log n)$. So the overall running time is

$$O(\log 1) + O(\log 2) + O(\log 3) + \ldots + O(\log n) = O(n \log n)$$

This is an upper bound. What is the lower bound?

- $\Omega(n)$
- $\Omega(n+1)$
- $\Omega(2^n)$
- $\Omega(n \log n)$

The lower bound trick – a less trivial example
We demonstrate this trick by giving an $\Omega(n \log n)$ bound on the time $T(n)$ required to insert $n$ keys into an (initially empty) balanced search tree.

The $i$'th insertion takes $K \log(i)$ time (for a constant $K$, that we ignore). Hence

$$\sum_{i=1}^{n} \log(i) = \log 1 + \log 2 + \log(\frac{n}{2}) + \log(\frac{n}{2} + 1) + \log(\frac{n}{2} + 2) + \ldots + \log(n)$$

This is an upper bound. What is the lower bound?

Random Variable (light version)
- Assume we perform an experiment (tossing a dice). Let $R$ be the result – one of the number 1, 2, 3, 4, 5, 6.
- We could define a random variable which (in this course) is a value that depends on the result of the experiment.
- Preferably, set to ‘1’ if some condition is satisfied, and is ‘0’ otherwise.
- Define $X$ to be a random variable, set to 1 if $R$ is even; $(X=0$ if $R$ is 1, 3, or 5).
- Define $Y$ to be another random variable, which is 1 if $R \geq 2$.
- We could ask what is the probability that $X=1$. Denote $Pr(X=1)$
- If dice is fair, $Pr(X=1)$ is 0.5, and $Pr(Y=1)=\frac{4}{6}=0.666$.

Random Variable and expectation (light version)
- In many cases, we would like to know what is the expected value of a random var.
- Example: If $Y=1$ we earn a dollar. What is the expected amount we earn in one game.
- Good news (for Boolean vars): $E(Y)$, the expected value of $Y$, is just $Pr(Y=1)$.
- What if we earn $17$ if $Y=1$.
- Lemma: For any constant $a$ it is always true that $E(aY) = aE(Y)$.

Analysis of QuickSort
- On whiteboard