CSCI 470 Fundamentals of Algorithms: Introduction, Insertion Sort

Vijay Chaudhary

April 24, 2023

Department of Electrical Engineering and Computer Science
Howard University

Overview

1. Introduction
2. Why are you here?
3. Algorithms as a technology
4. Insertion Sort
5. Proof by Induction (Optional)
6. Insertion Sort: Proof of Correctness
7. RAM Model of Computation
8. Insertion Sort: Runtime Analysis

Introduction

Course Information

- My contact info: Vijay Chaudhary, vijay.chaudhary.phd@gmail.com (temporarily)
- My office hours: Monday & Wednesday, 11 am - 12 pm at **TBA**.
- TA Office hours: TBA
- Textbook: *Introduction to Algorithms* by Cormen, Leiserson, Rivest, and Stein (Third Edition, MIT Press) ISBN: 9780262033848, which is commonly known as CLRS

Grading Policies: Quizzes

- Quizzes (5% of grade):
  - **∼10 quizzes**
  - There will be a quiz at a random time during lecture once a week, based on previous lectures.

Grading Policies: Homework

- Homework (45% of grade):
  - **6 homeworks** (+ Homework 0)

Grading Policies: Homework

- Homework (45% of grade):
  - **6 homeworks** (+ Homework 0)
  - Homeworks will be due in a week once they are released.

Grading Policies: Homework

- Homework (45% of grade):
  - **6 homeworks** (+ Homework 0)
  - Homeworks will be due in a week once they are released.
  - **No late submission will be accepted.** One of the homeworks with the lowest grade will be dropped. You should still attempt all of the assignments.

Grading Policies: Homework

- Homework (45% of grade):
  - **6 homeworks** (+ Homework 0)
  - Homeworks will be due in a week once they are released.
  - **No late submission will be accepted.** One of the homeworks with the lowest grade will be dropped. You should still attempt all of the assignments.
  - You may discuss approaches to solving a problem with others, however, your submission must be written on your own words. **Plagiarized submissions will be zero-ed out.**

Grading Policies: Homework

- Homework (45% of grade):
  - **6 homeworks** (+ Homework 0)
  - Homeworks will be due in a week once they are released.
  - **No late submission will be accepted.** One of the homeworks with the lowest grade will be dropped. You should still attempt all of the assignments.
  - You may discuss approaches to solving a problem with others, however, your submission must be written on your own words. **Plagiarized submissions will be zero-ed out.**
  - I will use Github temporarily to post lectures slides, notes, and assignments, until I get access to Canvas.

Grading Policies: Homework

- Homework (45% of grade):
  - **6 homeworks** (+ Homework 0)
  - Homeworks will be due in a week once they are released.
  - **No late submission will be accepted.** One of the homeworks with the lowest grade will be dropped. You should still attempt all of the assignments.
  - You may discuss approaches to solving a problem with others, however, your submission must be written on your own words. **Plagiarized submissions will be zero-ed out.**
  - I will use Github temporarily to post lectures slides, notes, and assignments, until I get access to Canvas.
  - Homework 0 has been posted on Github. You will receive extra credit points for this work. These are fairly easy problems for someone who has completed the prerequisites for this course.

Grading Policies: Exams

- Midterm Exams (30% of grade):

Grading Policies: Exams

- Midterm Exams (30% of grade):
  - There will be 2 midterm exams evenly spaced out during the semester. Exam 1 will be on **Wednesday, Sept. 20**, Exam 2 will be on **Monday, Oct. 23**.

Grading Policies: Exams

- Midterm Exams (30% of grade):
  - There will be 2 midterm exams evenly spaced out during the semester. Exam 1 will be on **Wednesday, Sept. 20**, Exam 2 will be on **Monday, Oct. 23**.
- Final Exam (20% of grade): Final Exam will be during the finals week. The schedule for the final exam will be scheduled by the department.

Why are you here?

Why are you here?

- Algorithms is fundamental to computer science.

Why are you here?

- Algorithms is fundamental to computer science.
- Algorithms is useful.

Why are you here?

- Algorithms is fundamental to computer science.
- Algorithms is useful.
- Algorithms is interesting and (probably) fun!

Algorithms as a technology

Algorithms as a technology

What if, we had infinitely fast computers and computer memory were free? Would we still need algorithms?

Algorithms as a technology

- The answer is **yes**. But why?

Algorithms as a technology

- The answer is **yes**. But why?
- We'd still need to make sure that a computer program would terminate and give us the desired output.

Algorithms as a technology

- The answer is **yes**. But why?
- We'd still need to make sure that a computer program would terminate and give us the desired output.
- However, we live in a world, where there are limited computing resources, and we need to make the best out of them.

Efficiency

Let's compare the efficiency of two sorting algorithms with different runtime complexities on a hardware.

Comparing Insertion Sort and Merge Sort

	Computer A	Computer B
Algorithm	Insertion Sort	Merge Sort
Computing Power	10 billion instructions per second	10 million instructions per second
Computer Program Quality	highly efficient	average
Runtime to sort n numbers	$2n^2$	$50n \lg n$
size of array; n	10 million = 10^7	10 million = 10^7
# of instructions to sort n numbers	$2 * (10^7)^2$	$50 * (10^7) * \lg(10^7)$
Sorting time in seconds	$\frac{2*(10^7)^2 \text{ instructions}}{10^{10} \text{ instructions/second}}$	$\frac{50*(10^7)*\lg(10^7) \text{ instructions}}{10^7 \text{ instructions/second}}$
	20,000 seconds	$\approx$ 1163 seconds
	5.56 hours	19.38 minutes

Efficiency

What if we have 100 million numbers to sort? Can you use the calculations above to find the runtime difference between insertion sort and merge sort for the computer A and computer B?

Class Exercise

What is the smallest value of n, say n_0, such that an algorithm (Algorithm 1) whose running time is $100n^2$ runs faster than an algorithm (Algorithm 2) whose running time is 2^n on the same machine?

Insertion Sort

Insertion Sort

- We will discuss insertion sort to discuss **loop invariant** and **proof of correctness** of an algorithm.

Insertion Sort

- We will discuss insertion sort to discuss **loop invariant** and **proof of correctness** of an algorithm.
- Insertion sort is something we intuitively use to sort items in small numbers.

Insertion Sort: Sorting Cards

Sorting a Dealt Cards

Once a set of n cards is dealt to us,

- we pick a card, which is sorted trivially.

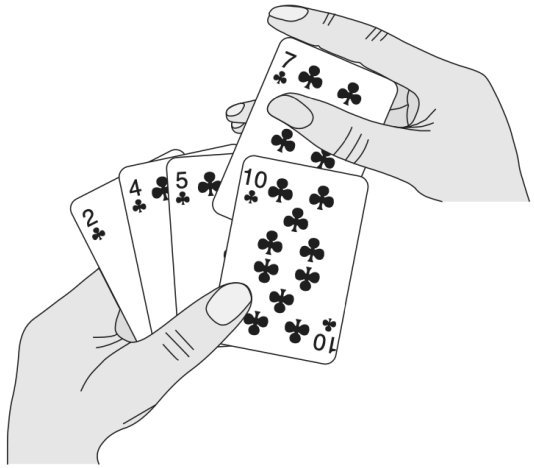

Figure 2.1 Sorting a hand of cards using insertion sort.

Insertion Sort: Sorting Cards

Sorting a Dealt Cards

Once a set of n cards is dealt to us,

- we pick a card, which is sorted trivially.
- as we continue picking the cards from the dealt pile on the table, we find the **insertion index** among the **sorted** set of cards on our hand to put the picked card.

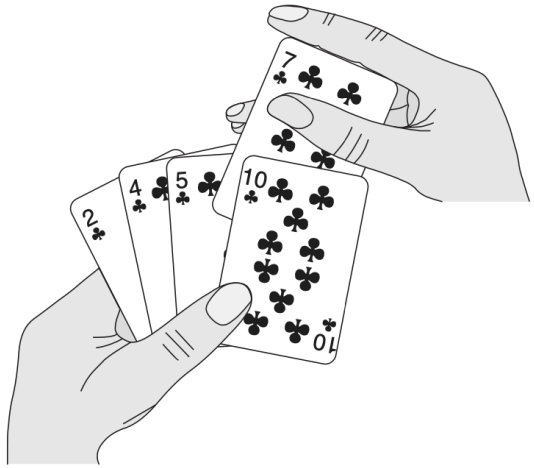

Figure 2.1 Sorting a hand of cards using insertion sort.

Insertion Sort: Sorting Cards

Sorting a Dealt Cards

Once a set of n cards is dealt to us,

- we pick a card, which is sorted trivially.
- as we continue picking the cards from the dealt pile on the table, we find the **insertion index** among the **sorted** set of cards on our hand to put the picked card.
- once we there is no more card left on the table to pick, we are left with a set of sorted cards on our hand.

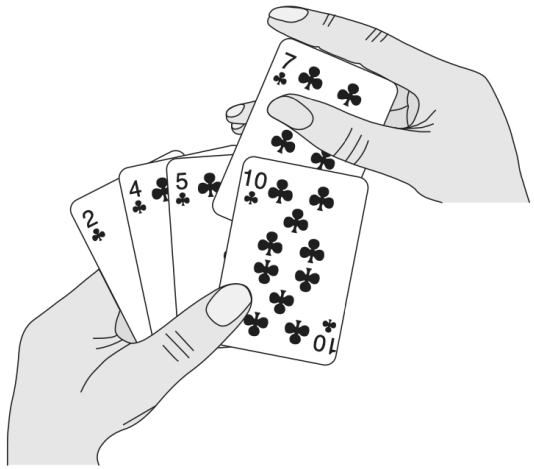

Figure 2.1 Sorting a hand of cards using insertion sort.

Insertion Sort: Formal Description

- Input: A sequence of n numbers $\langle a_1, a_2, a_3, ..., a_n \rangle$.
- Output: A permutation or reordering $\langle a_1', a_2', a_3', ..., a_n' \rangle$ of the input sequence such that $a_1' \leq a_2' \leq a_3' \leq ... \leq a_n'$.

INSERTION-SORT(A)

```
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1..j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

Insertion Sort: Illustration

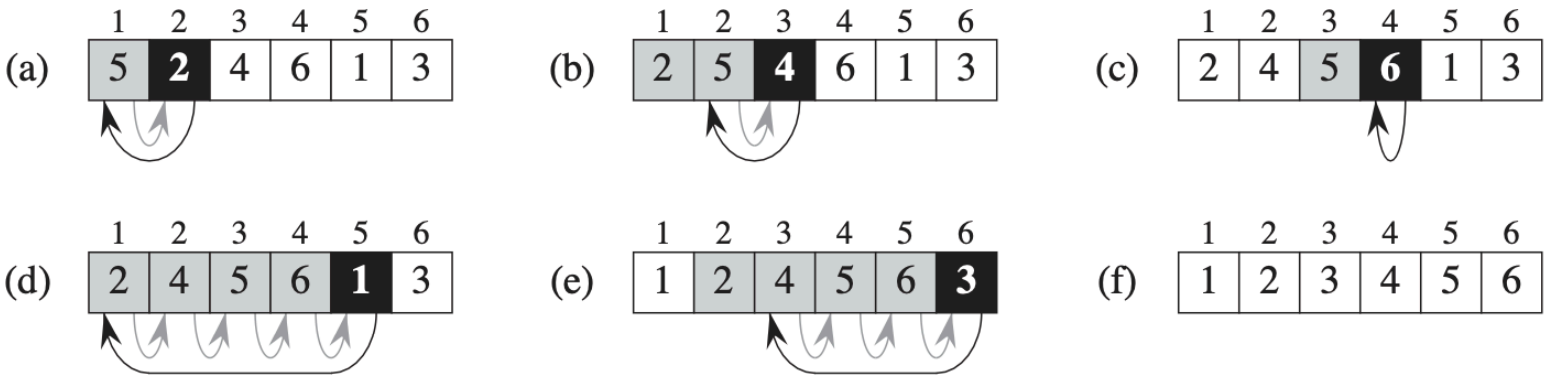

Figure 2.2 The operation of INSERTION-SORT on the array $A = \langle 5, 2, 4, 6, 1, 3 \rangle$. Array indices appear above the rectangles, and values stored in the array positions appear within the rectangles. **(a)–(e)** The iterations of the **for** loop of lines 1–8. In each iteration, the black rectangle holds the key taken from $A[j]$, which is compared with the values in shaded rectangles to its left in the test of line 5. Shaded arrows show array values moved one position to the right in line 6, and black arrows indicate where the key moves to in line 8. **(f)** The final sorted array.

Classwork

1. Using Figure 2.2 as a model, illustrate the operation of INSERTION-SORT on the array $A = \langle 31, 41, 59, 26, 41 \rangle$.
2. Rewrite the INSERTION-SORT procedure to sort into nonincreasing instead of non- decreasing order.

Insertion Sort: Loop Invariant

We will use **loop invariant** to write the **proof of correctness** for insertion sort.

- Why do you think we need to write a proof of correctness?

Insertion Sort: Loop Invariant

We will use **loop invariant** to write the **proof of correctness** for insertion sort.

- Why do you think we need to write a proof of correctness?
- What is a **loop invariant**?

Insertion Sort: Loop Invariant

We will use **loop invariant** to write the **proof of correctness** for insertion sort.

- Why do you think we need to write a proof of correctness?
- What is a **loop invariant**?
- Loop is ... you know what a loop is.

Insertion Sort: Loop Invariant

We will use **loop invariant** to write the **proof of correctness** for insertion sort.

- Why do you think we need to write a proof of correctness?
- What is a **loop invariant**?
- Loop is ... you know what a loop is.
- Invariant is something that does not change while going through a process/transformation.

Insertion Sort: Loop Invariant

We will use **loop invariant** to write the **proof of correctness** for insertion sort.

- Why do you think we need to write a proof of correctness?
- What is a **loop invariant**?
- Loop is ... you know what a loop is.
- Invariant is something that does not change while going through a process/transformation.
- Putting these together, a loop invariant is a statement or a condition that remains unchanged over iterations.

Loop Invariant

Writing a proof of correctness with a loop invariant needs three conditions.

- **Initialization**: The loop invariant is true prior to the first iteration in of the loop.

Note: These three conditions look very similar to how we write a proof by induction.

Loop Invariant

Writing a proof of correctness with a loop invariant needs three conditions.

- **Initialization**: The loop invariant is true prior to the first iteration in of the loop.
- **Maintenance**: If it is true before the i-th iteration of the loop, it remains true before the $(i+1)$-th iteration of the loop.

Note: These three conditions look very similar to how we write a proof by induction.

Loop Invariant

Writing a proof of correctness with a loop invariant needs three conditions.

- **Initialization**: The loop invariant is true prior to the first iteration in of the loop.
- **Maintenance**: If it is true before the i-th iteration of the loop, it remains true before the $(i+1)$-th iteration of the loop.
- **Termination**: When the loop terminates, the invariant gives us a useful property that helps us show the algorithm is correct.

Note: These three conditions look very similar to how we write a proof by induction.

Proof by Induction (Optional)

Proof by Induction (Optional)

Claim: The sum of n natural numbers, $n \geq 1$, is given by $S_n = \frac{n(n+1)}{2}$.

- For $n = 5$, $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5 = 15$

Proof by Induction (Optional)

Claim: The sum of n natural numbers, $n \geq 1$, is given by $S_n = \frac{n(n+1)}{2}$.

- For $n = 5$, $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5 = 15$
- Alternatively, $S_5 = \frac{5*(5+1)}{2} = 15$ (Holds for $n = 5$).

Proof by Induction (Optional)

Claim: The sum of n natural numbers, $n \geq 1$, is given by $S_n = \frac{n(n+1)}{2}$.

- For $n = 5$, $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5 = 15$
- Alternatively, $S_5 = \frac{5*(5+1)}{2} = 15$ (Holds for $n = 5$).
- For $n = 10$, $\sum_{i=1}^{10} = 1 + 2 + \ldots + 10 = 55$

Proof by Induction (Optional)

Claim: The sum of n natural numbers, $n \geq 1$, is given by $S_n = \frac{n(n+1)}{2}$.

- For $n = 5$, $\sum_{i=1}^{5} = 1 + 2 + 3 + 4 + 5 = 15$
- Alternatively, $S_5 = \frac{5*(5+1)}{2} = 15$ (Holds for $n = 5$).
- For $n = 10$, $\sum_{i=1}^{10} = 1 + 2 + ... + 10 = 55$
- Alternatively, $S_{10} = \frac{10*(10+1)}{2} = 55$ (Holds for $n = 10$).

Proof by Induction (Optional)

Can we prove that this claim holds for all $n \geq 1$ natural numbers?

Proof by Induction (Optional)

- **Base case:** $S_1 = 1$ (*True*)

Proof by Induction (Optional)

- **Base case:** $S_1 = 1$ (*True*)
- Inductive Step:

Proof by Induction (Optional)

- **Base case:** $S_1 = 1\,(True)$
- Inductive Step:
  - $S_k = \frac{k(k+1)}{2}$ where $1 \leq k \leq n$

Proof by Induction (Optional)

- **Base case:** $S_1 = 1$ (*True*)
- Inductive Step:
  - $S_k = \frac{k(k+1)}{2}$ where $1 \leq k \leq n$
  - $S_{k+1} = S_k + (k+1) = \frac{k(k+1)}{2} + (k+1) = (k+1)(\frac{k}{2}+1) = \frac{((k+1)((k+1)+1)}{2}$.

Proof by Induction (Optional)

- **Base case:** $S_1 = 1$ (*True*)
- Inductive Step:
  - $S_k = \frac{k(k+1)}{2}$ where $1 \leq k \leq n$
  - $S_{k+1} = S_k + (k+1) = \frac{k(k+1)}{2} + (k+1) = (k+1)(\frac{k}{2}+1) = \frac{((k+1)((k+1)+1)}{2}$.
  - $S_k \Rightarrow S_{k+1}$

Proof by Induction (Optional)

- **Base case:** $S_1 = 1\,(True)$
- Inductive Step:
  - $S_k = \frac{k(k+1)}{2}$ where $1 \leq k \leq n$
  - $S_{k+1} = S_k + (k+1) = \frac{k(k+1)}{2} + (k+1) = (k+1)(\frac{k}{2}+1) = \frac{((k+1)((k+1)+1)}{2}$.
  - $S_k \Rightarrow S_{k+1}$
- It follows by induction that $1 + 2 + ... + n = \frac{n(n+1)}{2}$ for all $n \geq 1, n \in \mathbb{N}$

Proof by Induction (Optional)

- **Base case:** $S_1 = 1$ ($True$) [**Initialization**]
- **Inductive Step:** [**Maintenance**]
  - $S_k = \frac{k(k+1)}{2}$ where $1 \leq k \leq n$
  - $S_{k+1} = S_k + (k+1) = \frac{k(k+1)}{2} + (k+1) = (k+1)(\frac{k}{2}+1) = \frac{((k+1)((k+1)+1)}{2}$.
  - $S_k \Rightarrow S_{k+1}$
- It follows by induction that $1 + 2 + ... + n = \frac{n(n+1)}{2}$ for all $n \geq 1, n \in \mathbb{N}$
- Unlike in mathematical induction where the "induction" runs infinitely, while writing proof with invariants, we stop the "induction" when the loop terminates.

Insertion Sort: Proof of Correctness

Pseudocode for Reference

```
INSERTION-SORT(A)
1  for j = 2 to A.length
2      key = A[j]
3      // Insert A[j] into the sorted sequence A[1..j − 1].
4      i = j − 1
5      while i > 0 and A[i] > key
6          A[i + 1] = A[i]
7          i = i − 1
8      A[i + 1] = key
```

Insertion Sort: Proof of Correctness

The **loop invariant** is a **sorted** subarray $A[1, \ldots, j-1]$ before the j-th iteration.

Initialization: In the pseudocode, the loop starts at $j = 2$. Therefore, before the loop starts, the subarray $A[1, \ldots, j-1]$ will be $A[1]$, which is sorted trivially, (before the j-th iteration, which is $j = 2$).

Insertion Sort: Proof of Correctness

Maintenance: Let's assume $A[1, \ldots, j-1]$ is sorted.

- In pseudocode for insertion sort above, lines $(4-7)$ is for finding the proper place to insert $A[j]$, and line 8 does the proper insertion such that $A[1, \ldots, j]$ is sorted.

Insertion Sort: Proof of Correctness

Maintenance: Let's assume $A[1, \ldots, j-1]$ is sorted.

- In pseudocode for insertion sort above, lines $(4-7)$ is for finding the proper place to insert $A[j]$, and line 8 does the proper insertion such that $A[1, \ldots, j]$ is sorted.
- Therefore, $A[1, \ldots, j]$ is sorted with the original elements from subarray $A[1, \ldots, j]$.

Insertion Sort: Proof of Correctness

Maintenance: Let's assume $A[1, \ldots, j-1]$ is sorted.

- In pseudocode for insertion sort above, lines $(4-7)$ is for finding the proper place to insert $A[j]$, and line 8 does the proper insertion such that $A[1, \ldots, j]$ is sorted.
- Therefore, $A[1, \ldots, j]$ is sorted with the original elements from subarray $A[1, \ldots, j]$.
- The loop invariant was sorted $A[1, \ldots, j-1]$ for $j-1$, and now it's sorted $A[1, \ldots, j]$ for j, ready for the next iteration.

Insertion Sort: Proof by Correctness

Termination: Now, let's look at what happens when the loop terminates.

- The loop terminates when $j > A.length$ and $A.length = n$, which happens when $j = n + 1$.

Therefore, $A[1, \ldots, n]$ is sorted, which is the required goal when the loop terminates.

Insertion Sort: Proof by Correctness

Termination: Now, let's look at what happens when the loop terminates.

- The loop terminates when $j > A.length$ and $A.length = n$, which happens when $j = n + 1$.
- As per the maintenance step, we can find that for $j = n + 1$, we will have $A[1, ..., j - 1] = A[1, ..., n + 1 - 1] = A[1, ..., n]$ sorted as that is the loop invariant.

Therefore, $A[1, ..., n]$ is sorted, which is the required goal when the loop terminates.

RAM Model of Computation

Assumptions

In this course, for most of the time we will use ***random-access machine (RAM)*** model of computation. Under this model of computation, we assume that the following instructions take constant time (borrowed from page 24 of CLRS):

- Arithmetic: add, subtract, multiply, divide, remainder, floor, ceiling
- Data movement: load, store, copy
- Flow control: conditional and unconditional branch, subroutine call, return

Insertion Sort: Runtime Analysis

Insertion Sort: Runtime Analysis

INSERTION-SORT(A)	*cost*	*times*
1 **for** $j = 2$ **to** $A.length$	c_1	n
2 $key = A[j]$	c_2	$n-1$
3 // Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
4 $i = j-1$	c_4	$n-1$
5 **while** $i > 0$ and $A[i] > key$	c_5	$\sum_{j=2}^{n} t_j$
6 $A[i+1] = A[i]$	c_6	$\sum_{j=2}^{n} (t_j - 1)$
7 $i = i-1$	c_7	$\sum_{j=2}^{n} (t_j - 1)$
8 $A[i+1] = key$	c_8	$n-1$

$$T(n) = c_1(n) + c_2(n-1) + c_4(n-1) + c_5 \sum_{j=2}^{n} t_j + c_6 \sum_{j=2}^{n} (t_j - 1) + c_7 \sum_{j=2}^{n} (t_j - 1) + c_8(n-1)$$

Insertion Sort: Best case analysis

In the best case, the inner while loop doesn't run. **Why though?**

$$
\begin{aligned}
T(n) &= c_1(n) + c_2(n-1) + c_4(n-1) + c_5\sum_{j=2}^{n} j + c_8(n-1) \\
&= c_1(n) + c_2(n-1) + c_4(n-1) + c_5(n-1) + c_8(n-1) \\
&= (c_1 + c_2 + c_4 + c_5 + c_8)n - (c_2 + c_4 + c_5 + c_8).
\end{aligned}
$$

Insertion Sort: Worst Runtime Analysis

In the worst case, the inner while loop run for j times at each iteration, which makes $t_j = j$. Then,

$$\sum_{j=2}^{n} j = \frac{n(n+1)}{2} - 1 \text{ and } \sum_{j=2}^{n} j - 1 = \frac{n(n-1)}{2}$$

$$\begin{aligned} T(n) &= c_1(n) + c_2(n-1) + c_4(n-1) + c_5\left(\frac{n(n+1)}{2} - 1\right) + \\ &\quad c_6\left(\frac{n(n-1)}{2}\right) + c_7\left(\frac{n(n-1)}{2}\right) + c_8(n-1) \\ &= \left(\frac{c_5}{2} + \frac{c_6}{2} + \frac{c_7}{2}\right)n^2 + \left(c_1 + c_2 + c_4 + \frac{c_5}{2} - \frac{c_6}{2} - \frac{c_7}{2} + c_8\right)n \\ &\quad - (c_2 + c_4 + c_5 + c_8) \end{aligned}$$

Insertion Sort: Average Runtime Analysis

An average case runtime analysis is based on the probability distribution of the inputs. If we assume that half of the items in subarray $A[1, \ldots, j-1]$ are less than the key, $A[j]$, and the rest of $A[1, \ldots, j-1]$ are greater. The inner while loop will run $t_j = j/2$ to drop the key at the proper insertion index. **I will leave this as an exercise to show, the average case runtime will still be quadratic function of the input size.**

Questions?