Algorithms & Data Structures I
CSC 225

Ali Mashreghi

Fall 2018

Department of Computer Science, University of Victoria

How good is the INSERTION-SORT algorithm?

- What does good even mean?!?!?!

- Does it mean easy to understand, fast, requiring little memory, having less power consumption, ….?

How good is the INSERTION-SORT algorithm?

- What does good even mean?!?!?!

- Does it mean easy to understand, fast, requiring little memory, having less power consumption, ….?

- In **this course** we consider **running time (speed)** as the main measure of **goodness** of algorithms.
- Sometimes we consider **memory consumption** as well.

How to compare running time?

How to compare running time?

- To compare the running time of two algorithms **A** and **B**:
  1. Implement both using the same language
  2. Provide them with the same input
  3. Run both programs on the same machine

How to compare running time?

- To compare the running time of two algorithms A and B:
  1. Implement both using the same programming language
     → This is a hassle
  2. Provide them with the same input
     → Providing one input doesn’t really seem to be enough
     → Moreover, even if we provide a set of inputs:
       (1) Does this set represent all possible inputs?
       (2) What if sometimes A is faster and sometimes B?
  3. Run both programs on the same machine
     → This may take quite some time

- This kind of comparison uses experimental results which is useful in its own way; however, we want all good things at the same time ☺

What is our ideal way of comparing algorithms?

- Comparing without implementing the algorithm
- Comparing without executing the code
- Comparing without considering every single input

Model of computation

- Similarities between the abstract mathematical world and the real world:

Real World	Abstract World
Program	Algorithm
Programming Language	Pseudocode
Computer What your program is allowed to do	????

Model of computation

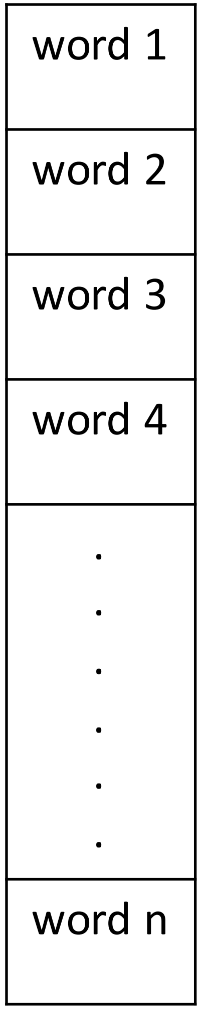
- Similarities between the abstract mathematical world and the real world:

Real World	Abstract World
Program	Algorithm
Programming Language	Pseudocode
Computer What your program is allowed to do	**Model of computation** **What your algorithm is allowed to do**

Model of computation

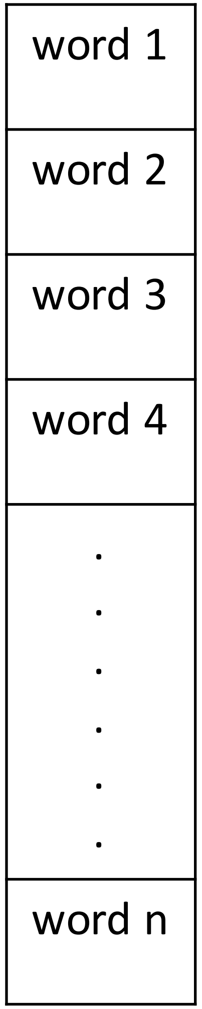
- Model of computation specifies what operations algorithm is allowed to do and the cost of each operation.
- We use Random Access Machine (RAM) as our model of computation.
- What else does RAM stand for?

Assumptions of RAM



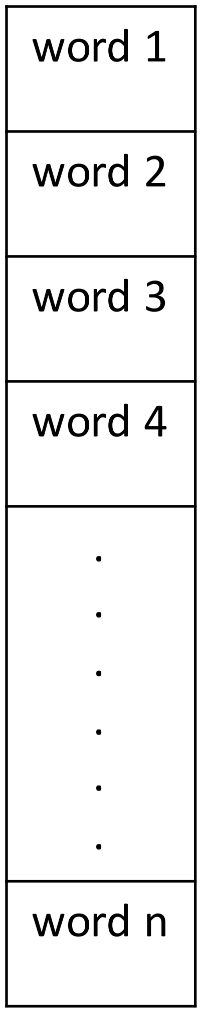
Assumptions of RAM

- RAM is actually very similar to Random Access Memory
- You can think of both of them as an **array** of **words**



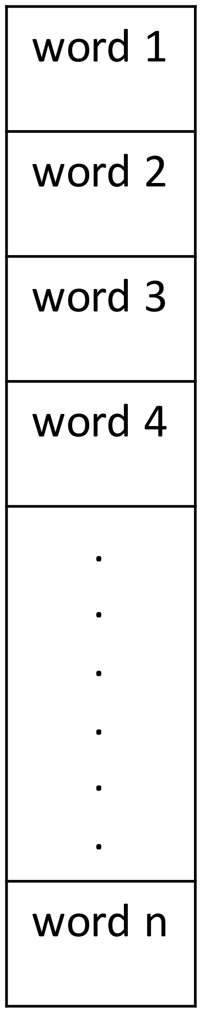
Assumptions of RAM

- RAM is actually very similar to Random Access Memory
- You can think of both of them as an **array** of **words**
- Because we assume **random access**, we can access and modify any location of this array in **one time unit**



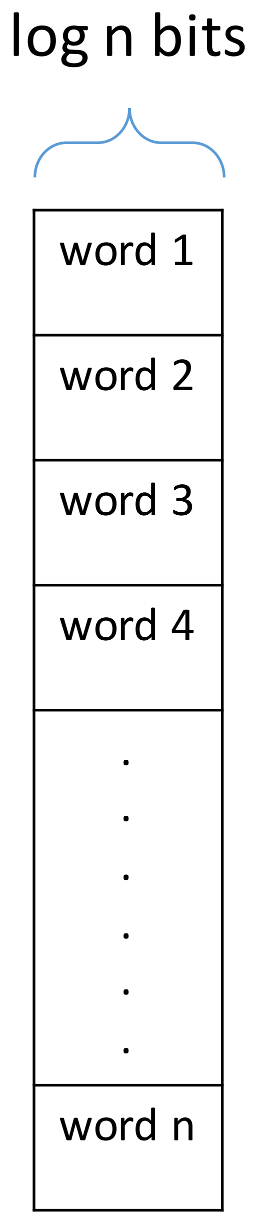
Assumptions of RAM

- RAM is actually very similar to Random Access Memory
- You can think of both of them as an **array** of **words**
- Because we assume **random access**, we can access and modify any location of this array in **one time unit**
- A **word** is a **unit of memory** that a computer uses. (w bits)



Assumptions of RAM

- RAM is actually very similar to Random Access Memory
- You can think of both of them as an **array** of **words**
- Because we assume **random access**, we can access and modify any location of this array in **one time unit**
- A **word** is a **unit of memory** that a computer uses. (w bits)
- We assume that when we are working with inputs of size n, w is at least $\log n$ bits. So, each word can hold the value of n. Ex. $n = 1024, w \geq\ 11$



Assumptions of RAM

1. Each simple operation (e.g. +, *, –, =, if, call) takes exactly one time step.

Assumptions of RAM

1. Each simple operation (e.g. +, *, –, =, if, call) takes exactly one time step.
2. Loops and subroutines are not considered simple operations. Instead, they are the composition of many single-step operations.

Assumptions of RAM

1. Each simple operation (e.g. +, *, –, =, if, call) takes exactly one time step.
2. Loops and subroutines are not considered simple operations. Instead, they are the composition of many single-step operations.
3. Each memory access takes exactly one time step. Further, we have as much memory as we need.

Assumptions of RAM

1. Each simple operation (e.g. +, *, –, =, if, call) takes exactly one time step.
2. Loops and subroutines are not considered simple operations. Instead, they are the composition of many single-step operations.
3. Each memory access takes exactly one time step. Further, we have as much memory as we need.

Note: It makes no sense for sort to be a single-step operation, since sorting 1,000,000 items will certainly take much longer than sorting 10 items.

Analysis of running time

- Assume size of the input array is n ($A.length = n$)

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
           sequence 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
```

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
          sequence 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
```

Type of operation	Required Time Units
assignment of j = 2	?
increment of j (addition)	?
field access on A.length (which is memory access)	?
comparison of j and A.length	?
Sum of all these operations	?

Analysis of running time

INSERTION-SORT(A)

```
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
        sequence 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
```

Type of operation	Required Time Units
assignment of j = 2	1
increment of j (addition)	n - 1
field access on A.length (which is memory access)	n (once when j == n+1)
comparison of j and A.length	n (once when j == n+1)
Sum of all these operations	3n

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
        sequence 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
```

Type of operation	Required Time Units
memory access A[j]	n - 1
assignment of key = A[j]	n - 1
Sum of all these operations	2n - 2

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
          sequence 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
```

Type of operation	Required Time Units
Nothing it's just a comment	0
Sum of all these operations	0

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
          sequence 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
```

Type of operation	Required Time Units
subtraction j - 1	n - 1
assignment i = j - 1	n - 1
Sum of all these operations	2n - 2

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
           sequence 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
```

Question: How many times is the while condition checked in terms of j?

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
           sequence 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
```

Question: How many times is the while condition checked in terms of j?

Answer: It depends on the value of key.

Analysis of running time

- Assume t_j is the number of times the while condition is checked for a specific j
  - For example, t_5 is the number of times the test of while loop is performed when $j = 5$

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
        sequence 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
```

Type of operation	Required Time Units
comparison i > 0	?
memory access A[i]	?
comparison of A[i] > key	?
evaluating the expression “i > 0 and A[i] < key”	?
Sum of all these operations	?

Analysis of running time

- Assume t_j is the number of times the while condition is checked for a specific j
  - For example, t_5 is the number of times the test of while loop is performed when $j = 5$

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
        sequence 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
```

Type of operation	Required Time Units
comparison i > 0	$t_2 + t_3 + \ldots + t_n = \sum_{j=2}^{n} t_j$
memory access A[i]	$\sum_{j=2}^{n} t_j$
comparison of A[i] > key	$\sum_{j=2}^{n} t_j$
evaluating the expression "i > 0 and A[i] < key"	$\sum_{j=2}^{n} t_j$
Sum of all these operations	$4\sum_{j=2}^{n} t_j$

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
          sequence 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
```

Type of operation	Required Time Units
memory access A[i]	$\sum_{j=2}^{n}(t_j - 1)$
memory access A[i+1]	$\sum_{j=2}^{n}(t_j - 1)$
addition of i + 1	$\sum_{j=2}^{n}(t_j - 1)$
assignment of A[i + 1] = A[i]	$\sum_{j=2}^{n}(t_j - 1)$
Sum of all these operations	$4\sum_{j=2}^{n}(t_j - 1)$

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
           sequence 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
```

Type of operation	Required Time Units
subtraction of i - 1	$\sum_{j=2}^{n}(t_j-1)$
assignment of i = i - 1	$\sum_{j=2}^{n}(t_j-1)$
Sum of all these operations	$2\sum_{j=2}^{n}(t_j-1)$

Analysis of running time

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
        sequence 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
```

Type of operation	Required Time Units
memory access A[i+1]	n - 1
addition of i + 1	n - 1
assignment of A[i + 1] = key	n - 1
Sum of all these operations	3n - 3

Analysis of running time

- The **running time of the algorithm** is the **sum** of running times **each operation** is executed

```
INSERTION-SORT(A)
for j = 2 to A.length
    key = A[j]
    // Insert A[j] into the sorted
        sequence 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
```

Required Time Units
3n
2n - 2
0
2n - 2
$4\sum_{j=2}^{n} t_j$
$4\sum_{j=2}^{n}(t_j - 1)$
$2\sum_{j=2}^{n}(t_j - 1)$
3n - 3
Running time of INSERTION-SORT = $10\sum_{j=2}^{n} t_j + 4n - 1$

Analysis of running time

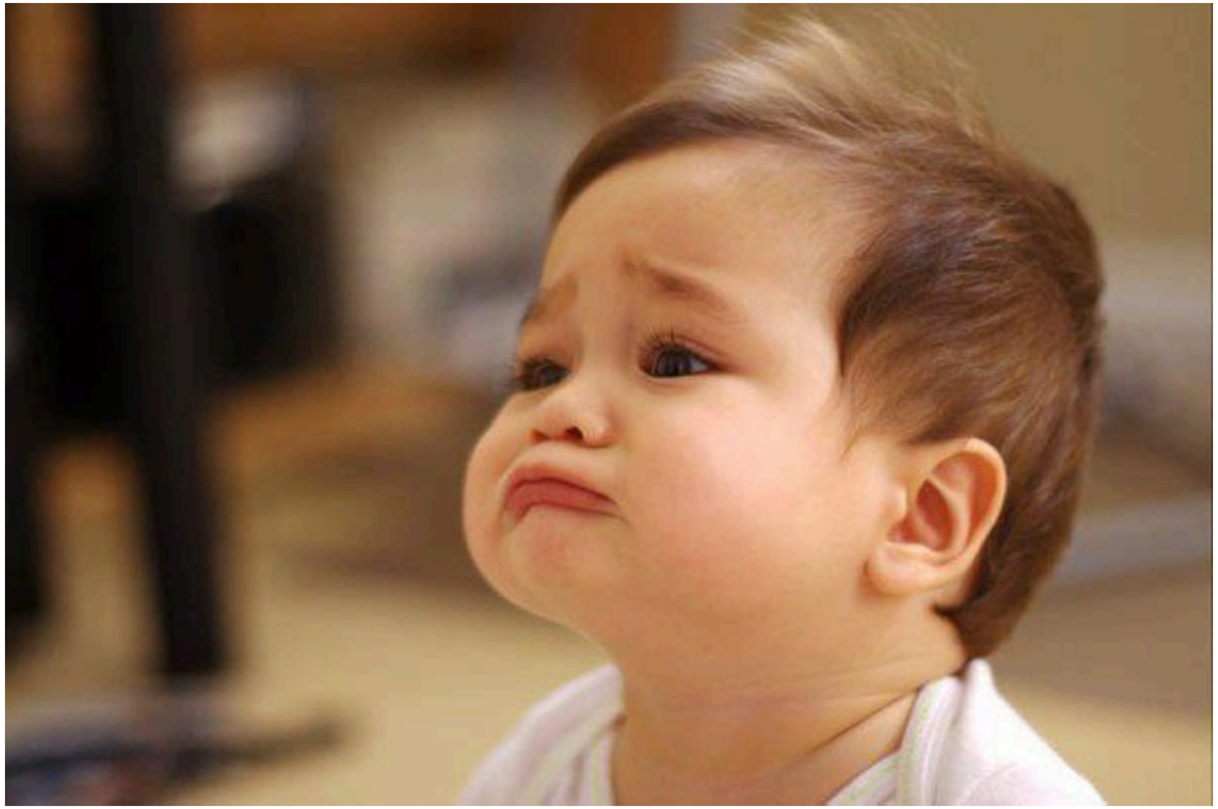
- Running time of INSERTION-SORT is

$$10\sum_{j=2}^{n} t_j + 4n - 1$$

Analysis of running time

- Running time of INSERTION-SORT is

$$10\sum_{j=2}^{n} t_j + 4n - 1$$



Analysis of running time

- Running time of INSERTION-SORT is

$$10\sum_{j=2}^{n} t_j + 4n - 1$$

- $t_2, \ldots, t_n$ depend on the input
- How can we fix it?

Analysis of running time

- Running time of INSERTION-SORT is

$$10\sum_{j=2}^{n} t_j + 4n - 1$$

- $t_2, \dots, t_n$ depend on the input
- How can we fix it?
    1. Consider the time on the **best input** (the one that causes the algorithm to work fastest) – This is called **best-case analysis**

Analysis of running time

- Running time of INSERTION-SORT is

$$10\sum_{j=2}^{n} t_j + 4n - 1$$

- $t_2, \ldots, t_n$ depend on the input
- How can we fix it?
  1. Consider the time on the **best input** (the one that causes the algorithm to work fastest) – This is called **best-case analysis**
  2. Consider the time on the **worst input** (the one that causes the algorithm to work slowest) – This is called **worst-case analysis**

Analysis of running time

- Running time of INSERTION-SORT is

$$10\sum_{j=2}^{n} t_j + 4n - 1$$

- $t_2, \ldots, t_n$ depend on the input
- How can we fix it?
  1. Consider the time on the **best input** (the one that causes the algorithm to work fastest) – This is called **best-case analysis**
  2. Consider the time on the **worst input** (the one that causes the algorithm to work slowest) – This is called **worst-case analysis**
  3. Consider the **average** time **on all inputs** – This is called **average-case analysis**