CSci 2011: Discrete Structures

Week 5

Sequences and Summations, Algorithms

Sequences and Summations

Sequences

- A sequence is a function from a subset of integers to a set S.
- The domain is usually {0,1, 2, ...} or {1,2,3, ...}
- Notation:
  - a_n : the image of integer n
  - $\{a_n\}$: entire sequence

Important Sequences $\{a_n\}$

- Arithmetic progression:
  - `a, a+d, a+2d, …, a+n*d, ….`
  - a is the initial term and d is the common difference
  - What is f(x) = dx + a?
  - Discrete analogue of a line
- Geometric progression
  - a, a*r, $a*r^2$, …, $a*r^n$, … $f(x) = ar^x$
  - Discrete analogue of the exponential function
  - a is the initial term and r is the common ratio

Find the formulae for these sequences

- 1, -1, 1, -1, …
  - $a_n = -1^n$, n = 0, 1, 2, …
  - Geometric sequence with a = 1 and r = -1

ar^n

- 1, ½, ¼,1/8, 1/16, …
  - $a_n = \frac{1}{2^n}$, n = 0, 1, 2, …
  - Geometric sequence with a = 1 and r = 1/2
- 1,3,5,7,9,..
  - $a_n = 2n + 1$, n = 0, 1, 2, …
  - Arithmetic series with a = 1 and d = 2
- 1,7,25,79,241,…
  - Compare with the terms for {3^n}: 3, 9, 27, 81, …
  - $a_n = 3^n - 2$, n = 1, 2, 3, …

$a + nd$

Find the formulae for these sequences

- 1, -1, 1, -1, …
  - $a_n = -1^n$, n = 0, 1, 2, …
  - Geometric sequence with a = 1 and r = -1
- 1, ½, ¼,1/8, 1/16, …
  - $a_n = \frac{1}{2^n}$, n = 0, 1, 2, …
  - Geometric sequence with a = 1 and r = 1/2
- 1,3,5,7,9,..
  - $a_n = 2n + 1$, n = 0, 1, 2, …
  - Arithmetic series with a = 1 and d = 2
- 1,7,25,79,241,…
  - Compare with the terms for {3^n}: 3, 9, 27, 81, …
  - $a_n = 3^n - 2$, n = 1, 2, 3, …

ar^n

$a + nd$

Recurrence Relations

Recurrence relations

- A recurrence relation for the sequence $\{a_n\}$ is an equation that expresses a_n in terms of previous elements in the sequence
- `1,2,4,8,16, …`
- $a_n = 2 * a_{n-1}$
- a_0 the initial term of the sequence needs to be given
- Sometimes we have two initial terms if the recurrence relation is identified in terms of a_{n-2}
- A sequence is called a solution of a recurrence relation if its terms satisfy the relation

Fibonacci Sequence

- $a_n = a_{n-1} + a_{n-2}$
- $a_0 = 0$, $a_1 = 1$

- Will be looked at in chapter 5

Solution to recurrence relations

- We like to find a solution to a recurrence relation in form of a sequence formula $\{a_n\}$ called a closed form or solution of the recurrence relation.

A practical example: compound interest

- If you put $10K in the bank with 11% annual (compound) interest, how much will there be in your account after 30 years?
- P_n = amount after n years
- $P_n = P_{n-1} + 0.11 P_{n-1}$
  $= 1.11 * P_{n-1}$
- $P_0 = 10{,}000$
- Can you solve this recurrence?

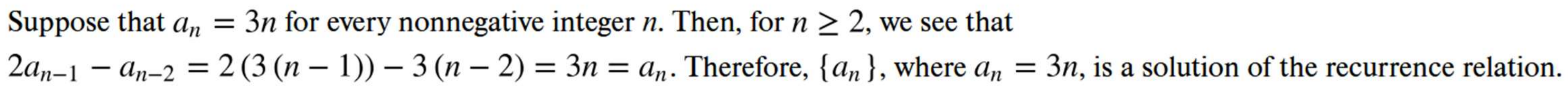
Answer

$$\begin{aligned}
P_1 &= (1.11)\,P_0 \\
P_2 &= (1.11)\,P_1 = (1.11)^2 P_0 \\
P_3 &= (1.11)\,P_2 = (1.11)^3 P_0 \\
&\vdots \\
P_n &= (1.11)\,P_{n-1} = (1.11)^n P_0 .
\end{aligned}$$

- $P_n = 1.11^n P_0$
- 10000 is the start value
- $P_n = 1.11^n * 10000$
- $P_{30} = 1.11^{30} * 10000 = \$228,923$

Example

- Is the sequence $\{a_n\}$
  $$a_n = 3n$$
- a solution for
  $$a_n = 2a_{n-1} - a_{n-2}$$
  $n=2,3,\ldots$


Suppose that $a_n = 3n$ for every nonnegative integer n. Then, for $n \geq 2$, we see that $2a_{n-1} - a_{n-2} = 2(3(n-1)) - 3(n-2) = 3n = a_n$. Therefore, $\{a_n\}$, where $a_n = 3n$, is a solution of the recurrence relation.

Exercise

- Find if the sequence $a_n = 4^n$ is a solution for recurrence relation $a_n = 8a_{n-1} - 16a_{n-2}$

Exercise

- Solve $a_n = a_{n-1} + 3$, $a_1 = 2$
- Verify that your solution is correct

$$\begin{aligned}
a_2 &= 2 + 3 \\
a_3 &= (2 + 3) + 3 = 2 + 3 \cdot 2 \\
a_4 &= (2 + 2 \cdot 3) + 3 = 2 + 3 \cdot 3 \\
&\vdots \\
a_n &= a_{n-1} + 3 = (2 + 3 \cdot (n - 2)) + 3 = 2 + 3\,(n - 1)\,.
\end{aligned}$$

- $a_n = 2 + 3(n-1)$

- Alternatively to solve $a_n = a_{n-1} + 3$, $a_1 = 2$
- Start with application of recurrence relation

$$\begin{aligned}
a_n &= a_{n-1} + 3 \\
&= (a_{n-2} + 3) + 3 = a_{n-2} + 3 \cdot 2 \\
&= (a_{n-3} + 3) + 3 \cdot 2 = a_{n-3} + 3 \cdot 3 \\
&\vdots \\
&= a_2 + 3\,(n-2) = (a_1 + 3) + 3\,(n-2) = 2 + 3\,(n-1)\,.
\end{aligned}$$

Solving recurrence relations

- As we have seen, a basic method to solve recurrence equation is to iteratively expand the equation

Exercise

- Find the solution for this recurrence relation by starting from the initial term:
- $a_n = 3a_{n-1}, a_0 = 2$

Summations

$$\sum_{i=m}^{n} a_i = a_m + a_{m+1} + \ldots + a_n$$

Summations

- It's just a convenient notation for regular addition

$$\sum_{i=1}^{n}(cx_i + dy_i) = c\sum_{i=1}^{n} x_i + d\sum_{i=1}^{n} y_i$$

Changing Summation Index

$$\sum_{i=1}^{5} e^{i} = \sum_{j=0}^{4} e^{j+1}$$

Some important summation formulae

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

Some important summation formulae

$$\sum_{i=0}^{n} r^{i} = \frac{r^{n+1}-1}{r-1}, r \neq 0,1$$

When |x| < 1

$$\sum_{i=0}^{\infty} x^i = \frac{1}{1-x}, |x| < 1$$

Other common notation

- Double summation

$$\sum_{i=0}^{1}\sum_{j=1}^{2}(i+j) = (0+1)+(0+2)+(1+1)+(1+2)$$

Set notation

- S = {3,5,7}

$$\sum_{x \in S} x = 3 + 5 + 7$$

Exercise

- Evaluate $\sum_{k=1}^{4} k^2$
- Evaluate $\sum_{k=1}^{4} 1$
- Evaluate $\sum_{k=1}^{4} (k^2 - 1)$

TABLE 2 **Some Useful Summation Formulae.**

Sum	*Closed Form*
$\sum_{k=0}^{n} ar^k \ (r \neq 0)$	$\frac{ar^{n+1} - a}{r - 1}, r \neq 1$
$\sum_{k=1}^{n} k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k, \lvert x\rvert < 1$	$\frac{1}{1-x}$
$\sum_{k=1}^{\infty} kx^{k-1}, \lvert x\rvert < 1$	$\frac{1}{(1-x)^2}$

Exercise

- Using the formulae, evaluate
- $\sum_{k=1}^{4} k^2$

- $\sum_{k=4}^{10} k^2$ noting that
- $\sum_{k=1}^{10} k^2 = \sum_{k=1}^{3} k^2 + \sum_{k=4}^{10} k^2$

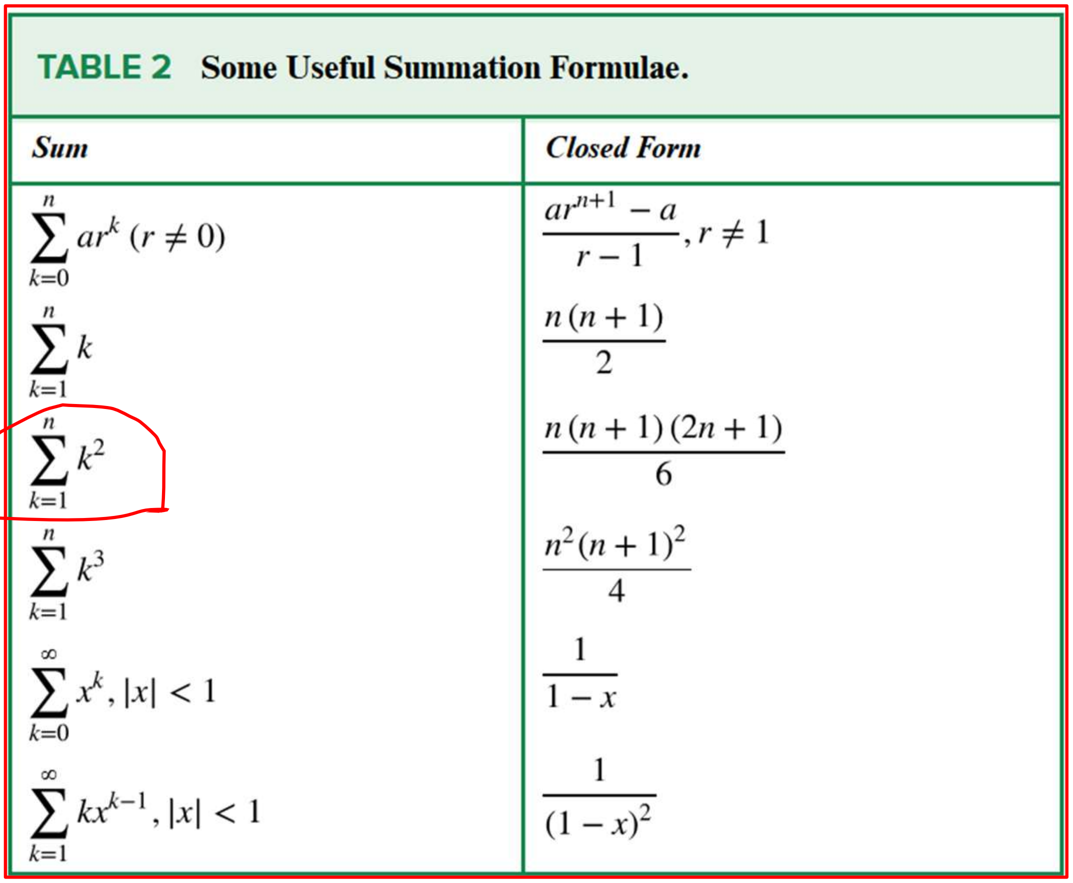


TABLE 2 Some Useful Summation Formulae.

Sum	*Closed Form*
$\sum_{k=0}^{n} ar^k \ (r \neq 0)$	$\frac{ar^{n+1} - a}{r - 1}, r \neq 1$
$\sum_{k=1}^{n} k$	$\frac{n(n+1)}{2}$
$\sum_{k=1}^{n} k^2$	$\frac{n(n+1)(2n+1)}{6}$
$\sum_{k=1}^{n} k^3$	$\frac{n^2(n+1)^2}{4}$
$\sum_{k=0}^{\infty} x^k, \lvert x\rvert < 1$	$\frac{1}{1-x}$
$\sum_{k=1}^{\infty} kx^{k-1}, \lvert x\rvert < 1$	$\frac{1}{(1-x)^2}$

Example

Find $\sum_{k=50}^{100} k^2$

$$\sum_{k=1}^{n} k^2 \qquad \frac{n(n+1)(2n+1)}{6}$$

$$\sum_{k=1}^{100} k^2 = \sum_{k=1}^{49} k^2 + \sum_{k=50}^{100} k^2$$

$$\sum_{k=50}^{100} k^2 = \sum_{k=1}^{100} k^2 - \sum_{k=1}^{49} k^2$$

$$\sum_{k=1}^{n} k^2 = n(n+1)(2n+1)/6$$

$$\sum_{k=50}^{100} k^2 = \frac{100 \cdot 101 \cdot 201}{6} - \frac{49 \cdot 50 \cdot 99}{6} = 338{,}350 - 40{,}425 = 297{,}925.$$

Practice

Compute the following sums:

$$\sum_{k=101}^{200} (10k - 5)$$

$$\sum_{j=1}^{z} \left(\sum_{k=1}^{z} jk \right)$$

Linear Combinations

$$\sum_{k=101}^{200} (10k - 5) = 10\left(\sum_{k=101}^{200} k\right) - \sum_{k=101}^{200} (5)$$

Linear Combinations

$$\sum_{k=101}^{200} (10k - 5) = 10\left(\sum_{k=101}^{200} k\right) - (100)(5)$$

Linear Combinations

$$\sum_{k=101}^{200} (10k - 5) = 10\left(\sum_{k=101}^{200} k\right) - 500$$

Linear Combinations

$$\sum_{k=101}^{200} (10k - 5) = 10(\sum_{k=1}^{200} (k) - \sum_{k=1}^{100} (k)) - 500$$

- $\sum_{k=m}^{z} a_j = (\sum_{k=m}^{p-1} a_j) + (\sum_{k=p}^{z} a_j)$ where $m < p < z$
- $\sum_{k=1}^{200} a_j = (\sum_{k=1}^{100} k) + (\sum_{k=101}^{200} k)$ where $0 < 101 < 200$

Linear Combinations

- $\sum_{k=101}^{200}(10k-5) = 10\sum_{k=101}^{200} k - \sum_{k=10}^{200}\ 5$

- $10\sum_{k=101}^{200} k = 10\left(\sum_{k=1}^{200} k - \sum_{k=1}^{100} k\right) = 10(\frac{200(201)}{2} - \frac{100(101)}{2})$

- $\sum_{k=101}^{200} 5 = 500$
- $\sum_{k=101}^{200}(10k-5) = 10(\frac{200(201)}{2} - \frac{100(101)}{2})\ -500 = 150000$

Double Summations

$$\sum_{j=1}^{z} \left(\sum_{k=1}^{z} jk \right)$$

Double Summations

$$\sum_{j=1}^{z} \left(\sum_{k=1}^{z} jk \right) = \sum_{j=1}^{z} \left(j \sum_{k=1}^{z} k \right)$$

Double Summations

$$\sum_{j=1}^{z} \left(\sum_{k=1}^{z} jk \right) = \sum_{j=1}^{z} \left(j \sum_{k=1}^{z} k \right) = \sum_{j=1}^{z} j \; \frac{(z)(z+1)}{2}$$

Double Summations

$$\sum_{j=1}^{z} \left(\sum_{k=1}^{z} jk \right) = \sum_{j=1}^{z} \left(j \sum_{k=1}^{z} k \right) = \sum_{j=1}^{z} j \frac{(z)(z+1)}{2} = \frac{(z)(z+1)}{2} \sum_{j=1}^{z} j$$

Double Summations

$$\sum_{j=1}^{z} \left(\sum_{k=1}^{z} jk \right) = \sum_{j=1}^{z} \left(j \sum_{k=1}^{z} k \right) = \sum_{j=1}^{z} j \quad \frac{(z)(z+1)}{2} = \frac{(z)(z+1)}{2} \sum_{j=1}^{z} j = \frac{(z)^2(z+1)^2}{4}$$

- We have concluded chapter 2
- The topic Matrices from chapter 2 will be introduced later in the semester closer to discussing data structure that rely on matrices

Chapter 3

- Algorithms

Algorithms

- Many problems involving discrete structures are solved using algorithms.
- An algorithm is a *finite* set of *precise* instructions for performing a computation that solves a problem
- Not too different from a proof

Let’s start with an example

- Given: a finite sequence of integers
- Output: maximum integer in the sequence

- We will specify algorithms using pseudocode: intermediate between plain English and a programming language

Max Algorithm

- Algorithm for finding the maximum element in a list:

```
**procedure** max ($a_1, a_2, \ldots, a_n$: integers)
*maxNum* := $a_1$
**for** i := 2 **to** n
  **if** maxNum < $a_i$ **then** *maxNum* := $a_i$
{return *maxNum* as the largest element}
```

Correctness

- Is the algorithm max correct?
- Does it produce the maximum value of n elements?
- With each iteration, we have the maximum of the first i-1 elements and we compare it with the i^{th} element and replace it with the new value if the new value is greater.
- At the n^{th} iteration, we compare the maximum value of the past n-1 elements with the new n^{th} value and adjust the max to be the value greater of the two.
- Max does hold the greatest value in the list of n element

Running Time of Algorithm Max

- Suppose we implemented this algorithm on a computer using a particular programming language and measured its running time in seconds.
- Which computer?
- Which language?
- Which input?

Running time

- To prevent our analysis from being system specific, we will simply count the number of “basic operations” as a function of the input size
- What are the basic operations for max?
    - Comparisons
- What is its running time?
    - The loop executes n-1 times from i=2 to i=n
    - This means n-1 comparisons

Max Algorithm

- Algorithm for finding the maximum element in a list:

```
**procedure** max ($a_1, a_2, \ldots, a_n$: integers)
$maxNum := a_1$
**for** i := 2 **to** n
  **if** maxNum < $a_i$ **then** $maxNum := a_i$
{return *maxNum* as the largest element}
```

A loop that runs n-1 times

Main operation is comparison

Common Properties of Algorithms

- Input: input values from a specified set S
- Output: for any input from S, the algorithm produces output values from a specified set
- Definiteness: the steps are defined precisely
- Correctness: should produce the correct output *for all input values*
- Finiteness: the number of steps should be finite
- Effectiveness: each step must be able to be performed exactly, and in a finite amount of time
- Generality: the algorithm should be applicable to all problems of a similar form

Max Algorithm

- Algorithm for finding the maximum element in a list:

```
procedure max ($a_1, a_2, \ldots, a_n$: integers)        input
$maxNum := a_1$
for i := 2 to n                                         finiteness
    if maxNum < $a_i$ then $maxNum := a_i$
{return maxNum as the largest element}                  output
```

Some Example Algorithm Problems

- Three classes of problems will be studied in this section.
  1. *Searching Problems*: finding the position of a particular element in a list.
  2. *Sorting problems*: putting the elements of a list into increasing order.
  3. *Optimization Problems*: determining the optimal value (maximum or minimum) of a particular quantity over all possible inputs.

Searching Algorithms

- Input: a sequence of n distinct integers $a_1, a_2, \ldots, a_n$; and an integer x
- Goal: to find if x is in the sequence
- Output: 0 if x is not in the sequence, the index of x otherwise

Linear Search Algorithm

```
**procedure** linear_search ($x$: integer; $a_1, a_2, \ldots, a_n$: integers)
$i := 1$
**while** ( $i \le n$ and $x \ne a_i$ )
    $i := i + 1$
**if** $i \le n$ **then** $location := i$
**else** $location := 0$
```

{*location* is the subscript of the term that equals x, or it is 0 if x is not found}

What is the running time of linear_search?

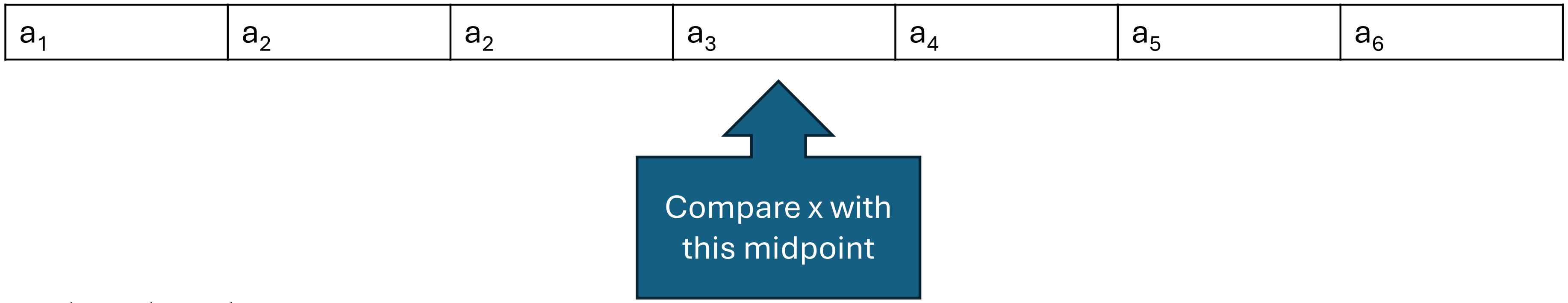
- Well, it depends not only the size of the input sequence, but also the location of x
- For such cases, often we use worst case analysis
- How many steps will linear_search take in the worst case?
- The worst case is when the number occurs at the end of the list making the algorithm iterate on all the elements of the list
- The worst case for the linear search is n iterations
- Can you do better than this in general?

Is there a better search algorithm?

Binary Search

- When the input is sorted, we can use this structure to find a more efficient search algorithm

Looking for x



$a_1 <= a_2 <= a_3 <= \ldots a_6$

If x is greater than the midpoint, then look in the right half of the list otherwise look in the left half of the list

Keep calculating the midpoint of the list that is shrinking as it narrows down on where x may be found

Binary Search

```
procedure binary_search ($x$: integer; $a_1, a_2, \ldots, a_n$: increasing integers)
$i$ := 1{ $i$ is left endpoint of search interval }
$j$ := $n${ $j$ is right endpoint of search interval }
while $i < j$
begin
    m := $\lfloor (i+j)/2 \rfloor$                    { $m$ is the point in the middle }
    if x > $a_m$ then $i$ := $m$+1
    else $j$ := $m$
end
if $x = a_i$ then location := $i$
else location := 0

{location is the subscript of x, or it is 0 if x is not found}
```

Search for **19** in the list:

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22,

i=1 j=16

Split the list

1 2 3 5 6 7 8 10 12 13 15 16 18 19 20 22.

i=9 j=16

Compare 19 with

12 13 15 16 18 19 20 22.

i=13 j=16

Compare 19 with

18 19 20 22.

i=13 j=14

Compare 19 with

18 19

i=14 j=14

Compare 19 with

19

Compare equality of 19.
Yes?
Return the index of 19
which is location 14

Running time of binary search

- After each guess, we reduce the set of possible locations by half
- $n, n/2, n/4, n/8, \ldots, n/2^i, \ldots$
- After the i^th step, we have a list of size $n/2^i$
- How many steps until we exhaust the list?
  $n/2^i = 1$, $2^i = n$
  - $i = \log(n)$
- Note: We often use log to denote logarithm base 2 (not base 10)

How significant is this improvement

n	log(n)
2	1
4	2
2^10 = 1024	10
2^20 ~ 1M	20
2^30 = 1,073,741,824 ~1Bn	30

Sorting

- If we will have many queries, it is definitely worth sorting the data
- This is called *preprocessing*
- In fact, sorting is one of the most common operations
- Algorithms we will consider: Insertion Sort and Bubble Sort