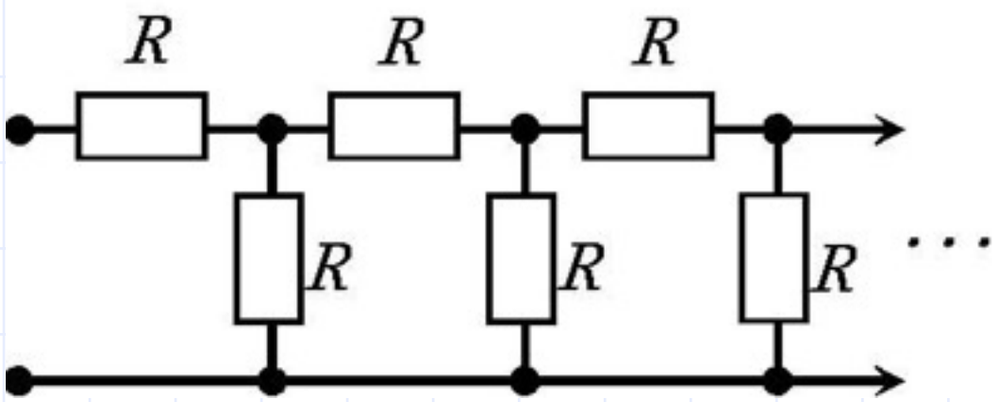
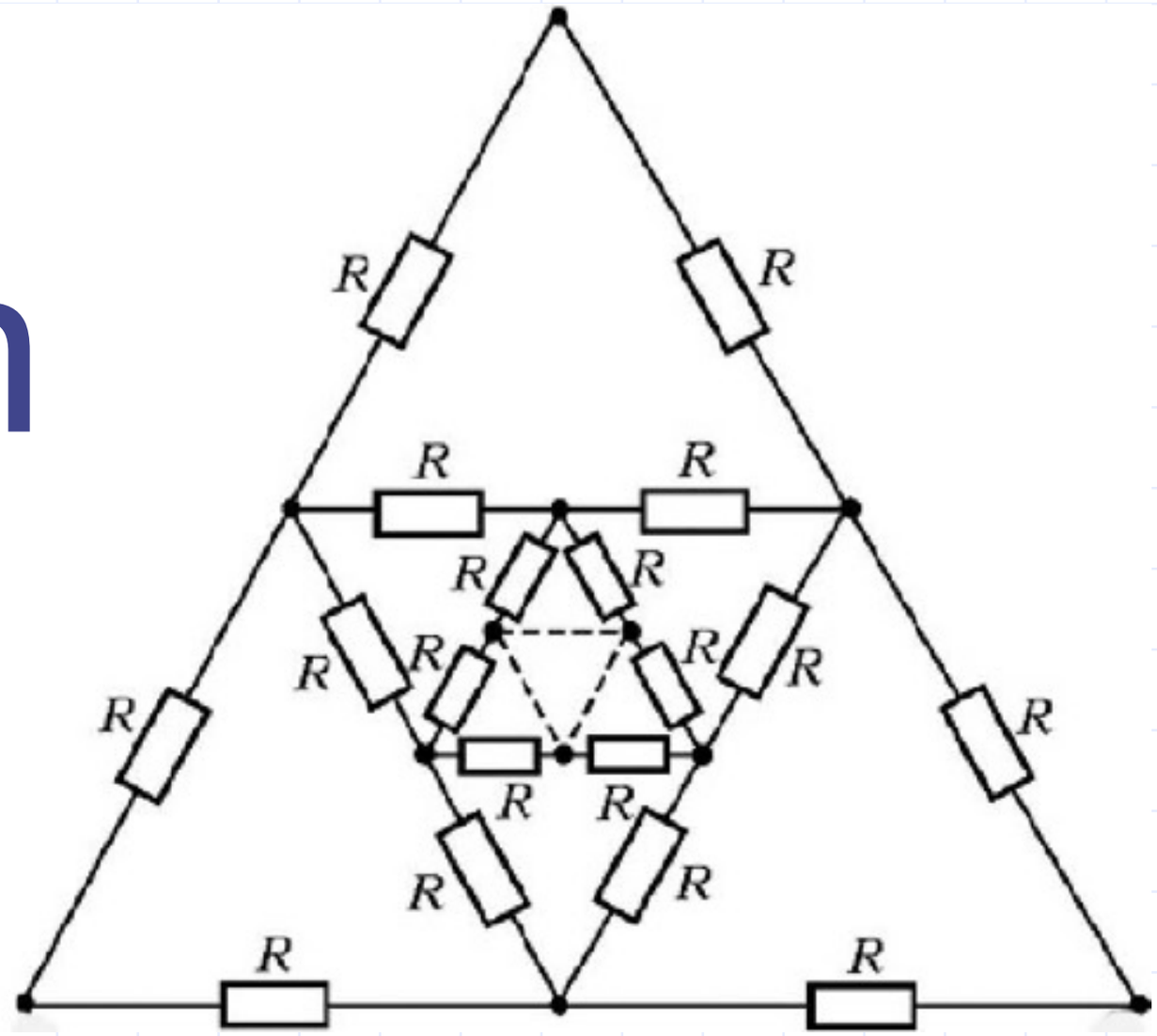
ESC101: Introduction to Computing

Recursion





Binary Search for Sorted Arrays

- binsearch(a, start, end, key)
  - Search key between a[start]…a[end], where a is a sorted (non-decreasing) array

```
if start > end, return 0;
mid = (start + end)/2 ;
if a[mid]==key, return 1;
if (a[mid] > key)
        return binsearch(a, start, mid-1, key);
else  return binsearch(a, mid+1, end, key);
```


It matters for the time taken!
How does it matter?

Estimating the Time taken

```
if start > end, return 0;
mid = (start + end)/2 ;
if a[mid]==key, return 1;
if (a[mid] > key)
        return binsearch(a, start, mid-1, key);
else  return binsearch(a, mid+1, end, key);
```

- binsearch
  - Recurrence?

$$T(n) = T(n/2) + C$$

  - Solution?

$$T(n) \propto \log n$$

- The worst case run time of binsearch is proportional to the log of the size of array
  - Much faster than Search/Search1/Search2 for large arrays
  - Remember: It works for sorted arrays

Recursion vs Iteration

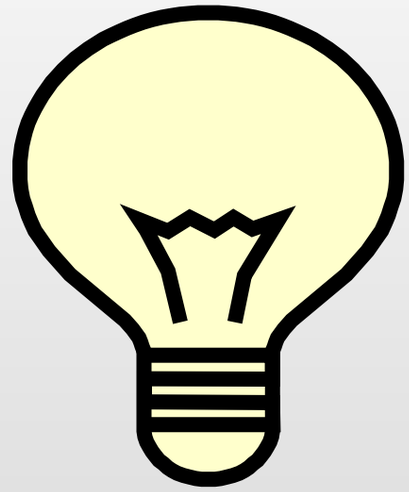
Fibonacci sequence: 0,1,1,2,3,5,8,13,…

```
int fib(int n)
{
   int first = 0, second = 1;
   int next, c;
   if (n <= 1)
       return n;
   for ( c = 1; c < n ; c++ ) {
       next = first + second;
       first = second;
       second = next;
    }
    return next;
}
```

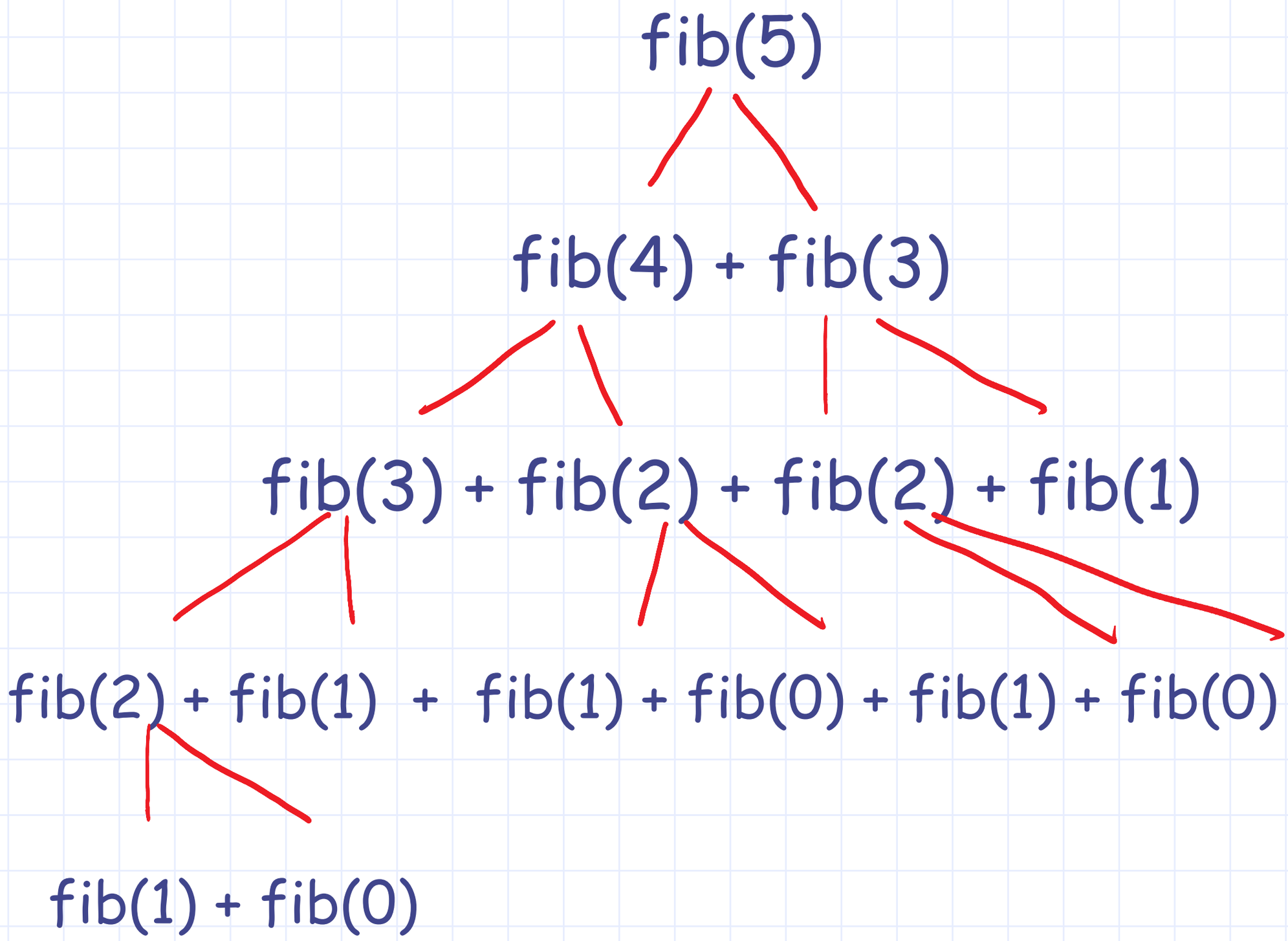
The recursive program is closer to the definition

But very very inefficient

```
int fib(int n)
{
   if ( n <= 1 )
      return n;
   else
      return fib(n-1) + fib(n-2);
}
```



Recursive Fibonacci



Recursive Fibonacci

```
#include<stdio.h>
int count = 0; /*Global: #fib calls */

int fib(int n) {
  count = count+1;
  if (n<=1) return n;
  else return fib(n-1) + fib(n-2);
}

int main() {
  int num, res;
  for (num=5; num<=30; num=num+5) {
    count = 0; /* reset the count*/
    res = fib(num);
    printf("%d, %d\n", res, count);
  }
  return 0;
}
```

num	fib(num)	Count
5	5	15
10	55	177
15	610	1973
20	6765	21891
25	75025	242785
30	832040	2692537

Recursion: Summary

- Advantages
  - Elegant. Solution is cleaner.
  - Fewer variables.
  - Once the recursive definition is figured out, program is easy to implement.
- Disadvantages
  - Debugging can be considerably more difficult.
  - Figuring out the logic of the recursive function is not easy sometimes.
  - Can be inefficient (requires more time and space), if not implemented carefully.

Tower of Hanoi

A B C

No disk may be placed on top of a smaller disk.

Image Source: http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

Recursion : Tower of Hanoi - 2

No disk may be placed on top of a smaller disk.

Image Source: http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

Recursion : Tower of Hanoi - 3

No disk may be placed on top of a smaller disk.

Image Source: http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

Recursion : Tower of Hanoi - 4

No disk may be placed on top of a smaller disk.

Image Source: http://www.comscigate.com/cs/IntroSedgewick/20elements/27recursion/index.html

```
// move n disks From A to C using B as auxx
void hanoi(int n, char A, char C, char B) {
  if (n==0) { return; } // nothing to move!!
   // recursively move n-1 disks
   // from A to B using C as auxx
   hanoi(n-1, A, B, C);
   // atomic move of a single disk
   printf("Move 1 disk from %c to %c\n", A, C);
   // recursively move n-1 disks
   // from B to C using A as auxx
   hanoi(n-1, B, C, A);
}
```

OUTPUT for hanoi(4, ‘A’, ‘C’, ‘B’)

Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C
Move 1 disk from A to B
Move 1 disk from C to A
Move 1 disk from C to B
Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C
Move 1 disk from B to A
Move 1 disk from C to A
Move 1 disk from B to C
Move 1 disk from A to B
Move 1 disk from A to C
Move 1 disk from B to C

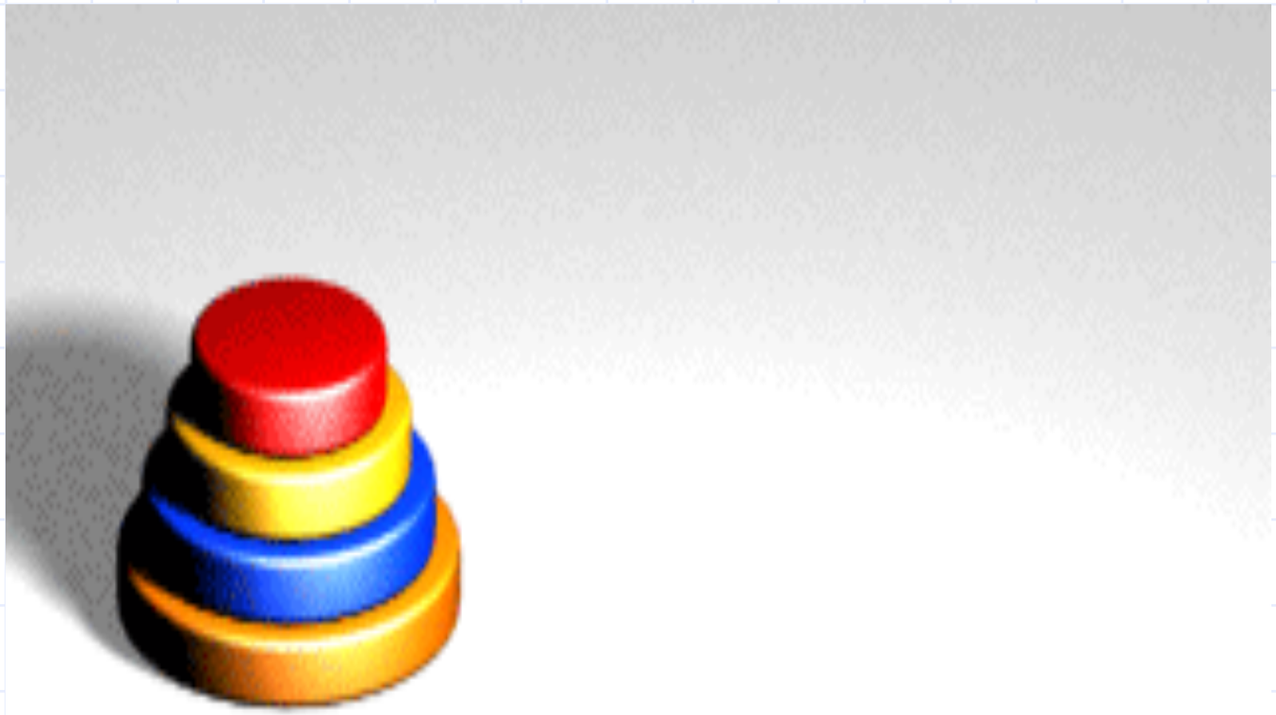


Image Source: http://upload.wikimedia.org/wikipedia/commons/6/60/Tower_of_Hanoi_4.gif

The puzzle was invented by the French mathematician **Édouard Lucas** in 1883.

There is a story about a temple in Kashi Vishwanath, which contains a large room with three posts surrounded by 64 golden disks. Brahmin priests have been moving these disks, in accordance with the immutable rules of the Brahma. The puzzle is therefore also known as the **Tower of Brahma** puzzle.

According to the legend, when the last move of the puzzle will be completed, the world will end.

If the legend were true, and if the priests were able to move disks at a rate of one per second, using the smallest number of moves, it would take them $2^{64}-1$ seconds or roughly 585 billion years or about 127 times the current age of the sun.

Source: https://en.wikipedia.org/wiki/Tower_of_Hanoi

ESC101: Introduction to Computing

Sorting

Sorting

- Given a list of integers (in an array), arrange them in ascending order.
  - Or descending order

INPUT ARRAY	5	6	2	3	1	4
OUTPUT ARRAY	1	2	3	4	5	6

- Sorting is an extremely important problem in computer science.
  - A common problem in everyday life.
  - Example:
    - Contact list on your phone.
    - Ordering marks before assignment of grades.

Sorted array have difficulty with

- inserting a new element while preserving the sorted structure.
- deleting an existing element while preserving the sorted structure.
- In both cases, there may be need to shift elements to the right or left of the index corresponding to insertion or deletion.

40	50	55	60	70	75	80	85	90	92

Example: Insert 65.
1. Find index where 65 needs to be inserted

40	50	55	60	65	70	75	80	85	90	92

2. Shift right from index 5 to create space.

3. Insert 65

May have to shift n-1 elements in the worst case.

Sorting

- Many well known sorting Algorithms
  - Selection sort
  - Merge sort
  - Quick sort
  - Bubble sort
  - ...
- Special cases also exist for specific problems/ data sets
- Different runtime
- Different memory requirements

Selection Sort

- Select the largest element in your array and swap it with the first element of the array.
- Consider the sub-array from the second element to the last, as your current array and repeat Step 1.
- Stop when the array has only one element.
  - Base case, trivially sorted

Selection Sort: Pseudo code

```
selection_sort(a[], start, end)  {
   if (start == end) /* base case, one elt => sorted */
       return;

   idx_max = find_idx_of_max_elt(a, start, end);
   swap(a, idx_max, start);
   selection_sort(a, start+1, end);
}
```

```
swap(a[], i, j)  {
   tmp = a[i];
   a[i] = a[j];
   a[j] = tmp;
}
```

```
main() {
   arr[] = { 5, 6, 2, 3, 1, 4 };
   selection_sort(arr, 0, 5);
   /* print arr */
}
```

Selection Sort: Properties

- Is the pseudo code iterative or recursive?
- What is the estimated run time when input array has **n** elements
  - for swap **Constant**
  - for find_idx_of_max_elt $\propto n$
  - for selection_sort **?**
- **Practice**: Write C code for iterative and recursive versions of selection sort.

```c
#include <stdio.h>

void swap(int a[], int i, int j)  {
   int tmp;
   tmp = a[i];
   a[i] = a[j];
   a[j] = tmp;
}

int find_idx_of_max_elt(int a[], int start, int end)
{
        int idx_max=start;
        for (int i=start+1; i<=end; i++)
                if(a[i]> a[idx_max])
                        idx_max = i;
        return idx_max;
}

void selection_sort( int a[], int start, int end)
{
        int idx_max;
        if(start == end)  /* base case, one elt => sorted */
                return;
        idx_max = find_idx_of_max_elt(a, start, end);
        swap(a, idx_max, start);
        selection_sort(a, start+1, end);
}

int main()
{
        int arr[]={10,100,4,20,45,56,72,43,33,86};
        for(int i=0; i<10; i++)
                printf("%d ",arr[i]);
        printf("\n");
        selection_sort(arr, 0, 9);
        for(int i=0; i<10; i++)
                printf("%d ",arr[i]);
        printf("\n");
        return 0;
}
```

Selection Sort: Time Estimate

- Recurrence

$$T(n) = T(n-1) + k_1 \times n + k_2$$

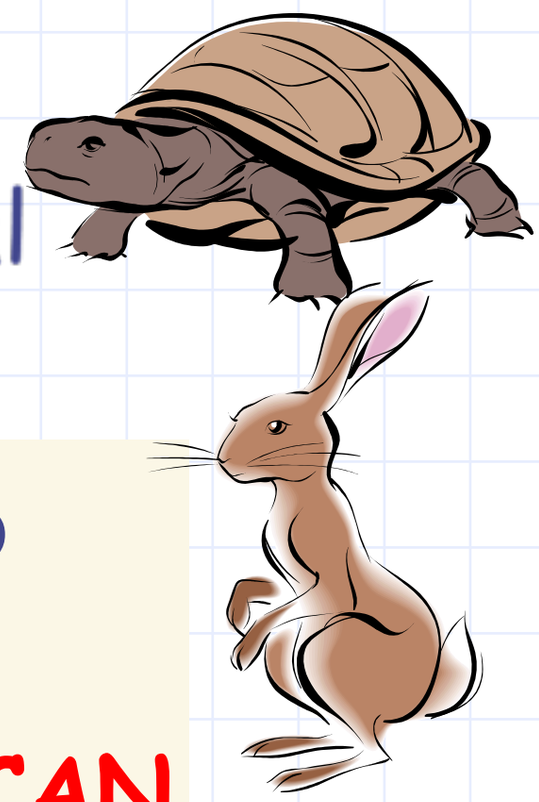
  - Solution

$$T(n) \propto n(n+1)$$

  - Or simply

$$T(n) \propto n^2$$

Selection sort runs in time proportional to the square of the size of the array to be sorted.



Can we do better?
YES WE CAN