Structs

Create our own data types via a struct. Syntax:

```
typedef struct
{
    string name;
    string number;
} person;
```

typedefin islenmesi:

```
// Implements a phone book with structs

#include <cs50.h>
#include <stdio.h>
#include <string.h>

typedef struct
{
    string name;
    string number;
} person;

int main(void)
{
    person people[3];

    people[0].name = "Yuliia";
    people[0].number = "+1-617-495-1000";

    people[1].name = "David";
    people[1].number = "+1-617-495-1000";

    people[2].name = "John";
    people[2].number = "+1-949-468-2750";

    // Search for name
    string name = get_string("Name: ");
    for (int i = 0; i < 3; i++)
    {
        if (strcmp(people[i].name, name) == 0)
        {
            printf("Found %s\n", people[i].number);
            return 0;
        }
    }
    printf("Not found\n");
    return 1;
```

```
}
```

Sort

Selection sort – burada ilk olaraq butun array gezilib en kicik axtarilir sonra 0 ci (i-ci) indexe getirilir. Sonra 1 den (novbeti i-den) baslayaraq yene butun array gezilir (amma bu defe n-1 step olur). Ve balaca axtarilib, 1-e getirilir.

```
For i from 0 to n-1
    Find smallest number between numbers[i] and numbers[n-1]
    Swap smallest number with numbers[i]
```

- Summarizing those steps, the first time iterating through the list took `n - 1` steps. The second time, it took `n - 2` steps. Carrying this logic forward, the steps required could be represented as follows:

  ```
  (n - 1) + (n - 2) + (n - 3) + ... + 1
  ```

- This could be simplified to n(n-1)/2 or, more simply, $O(n^2)$. In the worst-case or upper-bound, selection sort is in the order of $O(n^2)$. In the best-case, or lower-bound, selection sort is in the order of $\Omega(n^2)$.

Bu alqoritmde O(n^2) olmus olacaq (yuxaridaki sekile bax, orada /2 kimi ve n/2 kimi kiick ferq yaradan ededleri ciddiye almiriq), ve best scenario yene Ω(n^2) olacaq cunki, no matter what axira qeder check edir butun arrayi. Demeli Θ(n^2) olacaq. Cunki O ile maksimum operation gosterilir, Ω ile en yaxsi hal gosterilir, ve eger O ve Ω eyni olsa o zaman Θ da olur o eded.

Bubble sort – tekrar-tekrar 2-2 muqayise ede ede sort edir.

```
Repeat n-1 times
    For i from 0 to n-2
        If numbers[i] and numbers[i+1] out of order
            Swap them
    If no swaps
        Quit
```

Eger son conditionu anlamadinsa,
https://www.youtube.com/live/iCx3zwK8Ms8?si=ACVRM0aNj1Oi8c9J&t=5466

This way it still is O(n^2), but the lower bound can be Ω(n). therefore, it is more efficient than Selection sort.

Bubble Sort

- In bubble sort, the idea of the algorithm is to move higher valued elements generally towards the right and lower value elements generally towards the left.

In pseudocode:

- Set swap counter to a non-zero value
- Repeat until the swap counter is 0:
  - Reset swap counter to 0
  - Look at each adjacent pair
    - If two adjacent elements are not in order, swap them and add one to the swap counter

Merge sort

```
If only one number
    Quit
Else
    Sort left half of number
    Sort right half of number
    Merge sorted halves
```

O(nlog(n)) olmus olacaq burada, 2 hisseye bole bole sort olunur deye. Ω-de O ile bir olacaq. Yeni Θ da o eded olmus olur. N dene elementi log(n) defe divide and conquer edirik, ona gore de nlog(n) olmus scenariolar.

Runtime Analysis

Algorithm	O	Ω
Merge Sort	$O\ (Nlog(N))$	$\Omega\ (Nlog(N))$
Selection Sort	$O\ (N^2)$	$\Omega\ (N^2)$
Bubble Sort	$O\ (N^2)$	$\Omega\ (N)$

For example:

answers.txt

```
sort1 uses: Bubble Sort

How do you know?: on the reversed list, sort1 took 5s; on the sorted list, sort1 took .3s.

sort2 uses: Merge Sort

How do you know?: on the reversed list, sort2 took 0.275s; on the sorted list, 0.586s.

sort3 uses: Selection Sort

How do you know?: on the reversed list, sort3 took 2.4s; on the sourted list, 2.1s;
```

Recursion

Recursion – when a function calls itself, then it is recursive. Base case – ise recursive functionun sonudur. Meselen, binary searchde middle ile muqayise funksiyasi tekrar-tekrar davam edir, ve sonu start=end olanda olur, hansi ki base casedir.

Meselen 1cide mario piramidinin evvel yazdigimiz recursion olmayan usuludur. 2ci de ise recursion usulu

```c
// Draws a pyramid using iteration

#include <cs50.h>
#include <stdio.h>

void draw(int n);

int main(void)
{
    // Get height of pyramid
    int height = get_int("Height: ");

    // Draw pyramid
    draw(height);
}

void draw(int n)
{
    // Draw pyramid of height n
    for (int i = 0; i < n; i++)
    {
        for (int j = 0; j < i + 1; j++)
        {
            printf("#");
```

```
        }
        printf("\n");
    }
}
```

```c
// Draws a pyramid using recursion

#include <cs50.h>
#include <stdio.h>

void draw(int n);

int main(void)
{
    // Get height of pyramid
    int height = get_int("Height: ");

    // Draw pyramid
    draw(height);
}

void draw(int n)
{
    // If nothing to draw
    if (n <= 0)
    {
        return;
    }

    // Draw pyramid of height n - 1
    draw(n - 1);

    // Draw one more row of width n
    for (int i = 0; i < n; i++)
    {
        printf("#");
    }
    printf("\n");
}
```

Let's walk through your example with `n = 3`:

1. `draw(3)` is called. It calls `draw(2)`.
2. `draw(2)` is called. It calls `draw(1)`.
3. `draw(1)` is called. It calls `draw(0)`.
4. `draw(0)` is called. Since `n <= 0`, it returns immediately.

Now, the function starts to "come back" or "unwind" from the recursive calls:

5. `draw(1)` resumes after `draw(0)` returns. It draws one row of width 1.
6. `draw(2)` resumes after `draw(1)` returns. It draws one row of width 2.
7. `draw(3)` resumes after `draw(2)` returns. It draws one row of width 3.

Recursion

- The Collatz conjecture is applies to positive integers and speculates that it is always possible to get "back to 1" if you follow these steps:
  - If *n* is 1, stop.
  - Otherwise, if *n* is even, repeat this process on *n*/2.
  - Otherwise, if *n* is odd, repeat this process on 3*n* + 1.

- Write a recursive function `collatz(n)` that calculates how many steps it takes to get to 1 if you start from n and recurse as indicated above.

```
int collatz(int n)
{
// base case
if (n == 1)
return 0;
// even numbers
else if ((n % 2) == 0)
```

```
return 1 + collatz(n/2);
// odd numbers
else
return 1 + collatz(3*n + 1);
}
```

Yuxaridaki kodu anla sonra.