מבוא לشفת C

Tutorial 14: Recursion
Our plan for today

• Recursive merge of arrays
• Recursive search
• Recursive Sort
• Examples
Recursive merge
Remainder: Recursive Merge

• Receive two sorted arrays $a[\ ], b[\ ]$ and their sizes: $sizeA, sizeB$

• Return a single sorted array $c[\ ]$ of size $sizeA + sizeB$ which contains all the elements from $a[\ ]$ and from $b[\ ]$. 
Merging arrays: the recursive algorithm

- In each step, we’ll copy a single cell and call the recursive function again, without the cell we copied.
Merge – recursive implementation

```c
void merge(int a[], int sizeA, int b[], int sizeB, int c[])
{
    int takeFromA;
    if (sizeA == 0 && sizeB == 0) {
        return; /*both a and b are finished- our job is done */
    }

    if (sizeA == 0) { /*a is finished- copy cells from b */
        takeFromA = 0;
    } else if (sizeB == 0) {/*b is finished- copy cells from a */
        takeFromA = 1;
    } else { /* sizeB != 0 and sizeA != 0 */
        takeFromA = (a[0] < b[0]);
    }

    if (takeFromA) {
        c[0] = a[0];
        merge(a+1, sizeA-1, b, sizeB, c+1);
    } else { /* take from B */
        c[0] = b[0];
        merge(a, sizeA, b+1, sizeB-1, c+1);
    }
}
```
Recursive Search
Recursive Binary Search

- Lets implement a recursive binary search
- **Goal:** find the index of a cell whose value is X (If such cell exists)
- Return:
  - If such cell exists - return it’s index
  - Otherwise: return -1
Recursive Binary Search

1. Stopping conditions:
   a) If the size of the array is 0: return -1
   b) If the cell in the middle equals to our goal - return it’s index

2. If the cell in the middle is bigger than our goal:
   ▪ **Recursive step:** call the function again, but only with the first half of the array, and return the same

3. If the cell in the middle is smaller then our goal:
   ▪ **Recursive step:** call the function again, but only with the second half of the array. Return:
     ➢ If the recursive call returned -1: return the same
     ➢ Otherwise, add to the return value the amount of omitted cells (the first half).
Recursive Binary Search

```c
int binsearch(int a[], int size, int val)
{
    if (size == 0)
        return -1;
    if (a[size/2] == val)
        return size/2;
    if (a[size/2] > val)
        return binsearch(a, size/2, val);
    else {
        int pos = binsearch(a+(size/2)+1, size-(size/2)-1, val);
        if (pos == -1)
            return -1;
        return pos + size/2 + 1;
    }
}
```
Recursive Bubble Sort
## Remainder: bubble sort

<table>
<thead>
<tr>
<th>Iteration 1</th>
<th>Iteration 2</th>
<th>Iteration 3</th>
<th>Iteration 4</th>
</tr>
</thead>
<tbody>
<tr>
<td>7 9 2 5 3</td>
<td>7 2 5 3 9</td>
<td>2 5 3 7 9</td>
<td>2 3 5 7 9</td>
</tr>
<tr>
<td>7 9 2 5 3</td>
<td>7 2 5 3 9</td>
<td>2 5 3 7 9</td>
<td>2 3 5 7 9</td>
</tr>
</tbody>
</table>

The final sorted array is: 7 9 2 5 3.
Recursive Bubble Sort

- The idea:
  - Replace the outer loop with a recursion
  - As we all remember- in the end of each iteration the biggest value is already in the end of the array
  - Thus, in the next iteration, we can sort only what’s left from the array
  - Each recursive call we’ll pass as parameter only what's left from the array
  - **Stopping** condition will be an array of size 1 (because it’s already sorted).
• Our old friend:

```c
void swap(int *a, int *b) {
    int tmp = *a;
    *a = *b;
    *b = tmp;
}
```
Recursive Bubble Sort

```c
void rbs(int a[], int size)
{
    int i;
    if (size <= 1) {
        return; /* Already sorted */
    }

    for(i = 0; i < size-1; i++) {
        if (a[i] > a[i+1]) {
            swap(&a[i], &a[i+1]);
        }
    }

    rbs(a, size-1);
}
```

How can we replace this loop with a recursive function?
Execute the implementation on the example

```c
void rbs(int a[], int size) {
    int i;
    if (size <= 1) {
        return; /*Already sorted*/
    }
    for (i = 0; i < size - 1; i++) {
        if (a[i] > a[i+1]) {
            swap(&a[i], &a[i+1]);
        }
    }
    rbs(a, size-1);
}
```
What will happen if we switch the execution order in bubble sort?

```c
void rbs(int a[], int size)
{
    int i;
    if (size <= 1) {
        return; /* Already sorted */
    }

    rbs(a, size-1);

    for (i = 0; i < size-1; i++) {
        if (a[i] > a[i+1]) {
            swap(&a[i], &a[i+1]);
        }
    }
}
```
Execute the new implementation on the example:

- The new implementation is wrong.

Conclusions: the order is important.
Why didn’t it work?

- In order to understand why it didn’t work, we’ll try to figure out the logic of this program:
  - Array of size one is sorted - correct.
  - If we sort the array without its **last** cell, and then execute a single bubble iteration on the entire array. Will we receive a sorted array? **NO!**
  - For example:

```c
void rbs(int a[], int size)
{
    int i;
    if (size <= 1) {
        return; /*Already sorted*/
    }
    rbs(a, size-1);
    for (i = 0; i < size-1; i++) {
        if (a[i] > a[i+1]) {
            swap(&a[i], &a[i+1]);
        }
    }
}
```

Not sorted: 2 5 7 6 9

sorted: 2 5 7 9 6
Small fix

• Notice that the logic is correct if we add the new cell in the beginning

\[
\begin{array}{cccccc}
2 & 5 & 6 & 7 & 9 & \text{bubble}
\end{array}
\]

\[
\begin{array}{cccccc}
6 & 2 & 5 & 7 & 9 & \text{sorted}
\end{array}
\]

Also sorted

• So, if we change the program according to the new logic, it will work

```c
void rbs(int a[], int size)
{
    int i;
    if (size <= 1) {
        return; /*Already sorted*/
    }
    rbs(a+1, size-1);
    for (i = 0; i < size-1; i++) {
        if (a[i] > a[i+1]) {
            swap(&a[i], &a[i+1]);
        }
    }
}
```
GOOD LUCK