Studio 09-D

Sorting and memoization

Jia Xiaodong

22 Oct 2018

Loops

Definition

A for loop is made as such:

```
for (expr a; expr b; expr c) {
    statements
}
```

Definition

A while loop is made as such:

```
while (expression) {
    statements
}
```

Something useful

Loops model iterative processes.

Something useful

Loops model iterative processes.

Consider

```
function findmin(arr) {
    let a = Infinity;
    for (let i = 0; i < length(arr); i = i + 1) {
        if (arr[i] < a) {
            a = arr[i];
        }
    }
    return a;
}
```

Checking for existence

Consider

```
function find(arr, x) {
    for (let i = 0; i < length(arr); i = i + 1) {
        if (arr[i] === x) {
            return i;
        }
    }
    return -1;
}
```

Checking for existence

Consider

```
function find(arr, x) {
    for (let i = 0; i < length(arr); i = i + 1) {
        if (arr[i] === x) {
            return i;
        }
    }
    return -1;
}
```

Questions to ask:

- How fast is this?
- Can we do better?

Sorting

Easy! Consider this:

Sorting

Easy! Consider this:

Algorithm

Input: `arr` to sort.

1. If `arr` not empty, do `m = findMin(arr)`, remove `m` from `arr` (or set it to ∞).
2. Append `m` to an array `res`, return to step 1.
3. Return `res`.

Sorting

Easy! Consider this:

Algorithm

Input: `arr` to sort.

1. If `arr` not empty, do `m = findMin(arr)`, remove `m` from `arr` (or set it to ∞).
2. Append `m` to an array `res`, return to step 1.
3. Return `res`.

- How fast is this?
- Can we improve?

Better sorts

Merge sort:

- Split arrays into two

Better sorts

Merge sort:

- Split arrays into two
- Sort both halves recursively

Better sorts

Merge sort:

- Split arrays into two
- Sort both halves recursively
- Merge both halves in linear time

Better sorts

Merge sort:

- Split arrays into two
- Sort both halves recursively
- Merge both halves in linear time
- Merge at each level takes $O(n)$ time. There are $\lg n$ levels of splits. Total: $O(n \lg n)$ time.

Aside

I want to get the median element of an array.

- Brute force method: `findMin` $\frac{n}{2}$ times?

Aside

I want to get the median element of an array.

- Brute force method: `findMin` $\frac{n}{2}$ times?
  - How fast is this?

Aside

I want to get the median element of an array.

- Brute force method: `findMin` $\frac{n}{2}$ times?
  - How fast is this?
- Sort the array and take the $\frac{n}{2}$-th element?

Aside

I want to get the median element of an array.

- Brute force method: `findMin` $\frac{n}{2}$ times?
  - How fast is this?
- Sort the array and take the $\frac{n}{2}$-th element?
  - How fast is this?

Aside

I want to get the median element of an array.

- Brute force method: `findMin` $\frac{n}{2}$ times?
  - How fast is this?
- Sort the array and take the $\frac{n}{2}$-th element?
  - How fast is this?
- Can we go faster?

Select-Kth

```
function pivot(arr)...
function partition(arr, p)...
function select(k, arr) {
    p = pivot(arr);
    L, R = partition(arr, p);
    if (length(L) === k - 1)
        return arr[p];
    else if (length(L) > k - 1)
        return select(L, k);
    else if (length(L) < K - 1)
        return select(R, k - length(L) - 1);
}
```

Quicksort

```
function pivot(arr)...
function partition(arr, p)...
function quicksort(arr, lo, hi) {
    if (lo < hi) {
        const pivot = partition(A, lo, hi);
        quicksort(arr, lo, pivot - 1);
        quicksort(arr, pivot + 1, hi);
    }
}
```

Memoization

Memoization:

- Put things down on a memo pad.
- *Referentially transparent* [1].

[1] Usually you won't have to worry about this.

S10 Q1

Draw the environment during the evaluation of the following:

```
function swap(A, i, j) {
    let temp = A[i];
    A[i] = A[j];
    A[j] = temp;
}
function reverse_array(A) {
    const len = array_length(A);
    const half_len = math_floor(len / 2);
    let i = 0;
    while (i < half_len) {
        const j = len - 1 - i;
        swap(A, i, j);
        i = i + 1;
    }
}
const arr = [1, 2, 3, 4, 5];
reverse_array(arr);
arr;
```

Skipped

S10 Q2

```
function bubblesort_array(A) {
    const len = array_length(A);
    for (let i = len - 1; i >= 1; i = i - 1) {
        for (let j = 0; j < i; j = j + 1) {
            if (A[j] > A[j + 1]) {
                const temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            } else { }
        }
    }
}
```

What is the time complexity for this function?

S10 Q2

```
function bubblesort_array(A) {
    const len = array_length(A);
    for (let i = len - 1; i >= 1; i = i - 1) {
        for (let j = 0; j < i; j = j + 1) {
            if (A[j] > A[j + 1]) {
                const temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            } else { }
        }
    }
}
```

What is the time complexity for this function?

$O(n^2)$.

S10 Q2

Write `bubblesort_list` that works on lists instead of arrays.

S10 Q2

Write `bubblesort_list` that works on lists instead of arrays.

```
function bubblesort_array(A) {
    const len = array_length(A);
    for (let i = len - 1; i >= 1; i = i - 1) {
        for (let j = 0; j < i; j = j + 1) {
            // swap if larger
            if (A[j] > A[j + 1]) {
                const temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            } else { }
        }
    }
}
```

S10 Q2

Write `bubblesort_list` that works on lists instead of arrays.

```
function bubblesort_array(A) {
    const len = array_length(A);
    for (let i = len - 1; i >= 1; i = i - 1) {
        for (let j = 0; j < i; j = j + 1) {
            // swap if larger
            if (A[j] > A[j + 1]) {
                const temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            } else { }
        }
    }
}
```

```
function bubblesort_list(L) {
    const len = length(L);
    for (let i = len - 1; i >= 1; i = i - 1) {









    }
}
```

S10 Q2

Write `bubblesort_list` that works on lists instead of arrays.

```
function bubblesort_array(A) {
    const len = array_length(A);
    for (let i = len - 1; i >= 1; i = i - 1) {
        for (let j = 0; j < i; j = j + 1) {
            // swap if larger
            if (A[j] > A[j + 1]) {
                const temp = A[j];
                A[j] = A[j + 1];
                A[j + 1] = temp;
            } else { }
        }
    }
}
```

```
function bubblesort_list(L) {
    const len = length(L);
    for (let i = len - 1; i >= 1; i = i - 1) {
        let p = L;
        for (let j = 0; j < i; j = j + 1) {
            if (head(p) > head(tail(p))) {
                const temp = head(p);
                set_head(p, head(tail(p)));
                set_head(tail(p), temp);
            } else { }
            p = tail(p);
        }
    }
}
```

S10 Q3

```
function coin_change(amount, kinds_of_coins) {
    if (amount === 0) {
        return 1;
    } else if (amount < 0 || kinds_of_coins === 0) {
        return 0;
    } else {
        return coin_change(amount, kinds_of_coins - 1)
            + coin_change(amount - first_denomination(kinds_of_coins), kinds_of_coins);
    }
}
function first_denomination(kinds_of_coins) {
    return [undefined, 5, 10, 20, 50, 100][kinds_of_coins];
}
```

Can this be memoized?

S10 Q3

```
function coin_change(amount, kinds_of_coins) {
    if (amount === 0) {
        return 1;
    } else if (amount < 0 || kinds_of_coins === 0) {
        return 0;
    } else {
        return coin_change(amount, kinds_of_coins - 1)
            + coin_change(amount - first_denomination(kinds_of_coins), kinds_of_coins);
    }
}
function first_denomination(kinds_of_coins) {
    return [undefined, 5, 10, 20, 50, 100][kinds_of_coins];
}
```

Can this be memoized?

S10 Q3

Implement the memoized version, and give its space and time complexities.

S10 Q3

Implement the memoized version, and give its space and time complexities.

```
function mcc(n, k) {
    if (read(n, k) !== undefined) {
        return read(n, k);
    } else {
        const result = n === 0
            ? 1
            : n < 0 || k === 0
                ? 0
                : mcc(n, k - 1)
                    + mcc(n - first_denomination(k), k);
        write(n, k, result);
        return result;
    }
}
```