1 Revisiting Growth of Functions (CLRS §3.1)

Goal: Establish notation that enables us to compare relative performance of different algorithms.

Definition

- $T(n) = \mathcal{O}(g(n))$ means there exists $c > 0$ such that $T(n) \leq cg(n)$ for sufficiently large n.
- $T(n) = \Omega(g(n))$ means there exists $c > 0$ such that $T(n) \geq cg(n)$ for sufficiently large n.
- $T(n) = \Theta(g(n))$ means there exists c_1, c_2 such that $c_1 g(n) \leq T(n) \leq c_2 g(n)$ for sufficiently large n.
- $T(n) = o(g(n))$ means $\lim_{n\to\infty} \frac{T(n)}{g(n)} =$

Example $3n^3 + 5n^2 + 10643n \in \Theta(\qquad)$

Example Give an example of $T(n)$ and $g(n)$ such that $T(n) \neq o(g(n))$, but $T(n) = \mathcal{O}(g(n))$.

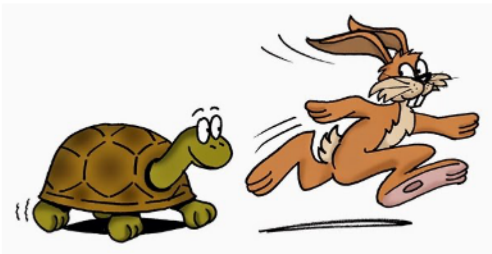
Example $5n^2 + 11 \in o(\qquad)$

1.1 Asymptotic notation in equations

A set in a formula represents an anonymous function in that set.

Example $f(n) = n^3 + \mathcal{O}(n^2)$

Example $n^2 + \mathcal{O}(n) = \mathcal{O}(n^2)$



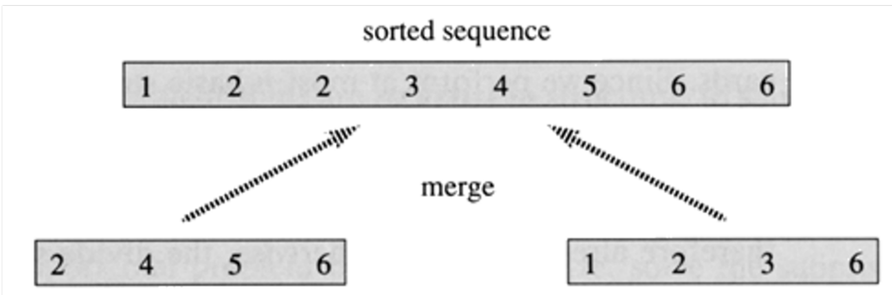
1.2 Proofs involving order of growth

Example

Claim. $f \in \mathcal{O}(g(n))$ if and only if $g \in \Omega(f(n))$
Proof.

2 Divide and Conquer

2.1 Mergesort (CLRS §2.3)



2.1.1 The Merge Subroutine

```
i = 1
j = 1
for k = 1 to
    if
        C[k] =
        i =
    else
        C[k] =
        j =
```

Alternative pseudocode:

```
To merge sorted arrays L[1 ... m] and R[1 ... p] into array C[1 ... m+p]
    Maintain a current index for each list, each initialized to 1
    While both lists have not been completely traversed:
        Let L[i] and R[j] be the current elements
        Copy the smaller of L[i] and R[j] to C
        Advance the current index for the array from which the smaller element
           was selected
    EndWhile
    Once one array has been completely traversed, copy the remainder of the
       other array to C
```

2.1.2 Proof of Correctness of Merge

Loop invariant:

- Initialization:

- Maintenance:

- Termination:

2.1.3 Running Time of Merge

2.1.4 Back to Mergesort: Correctness, Running Time, Recursion Tree

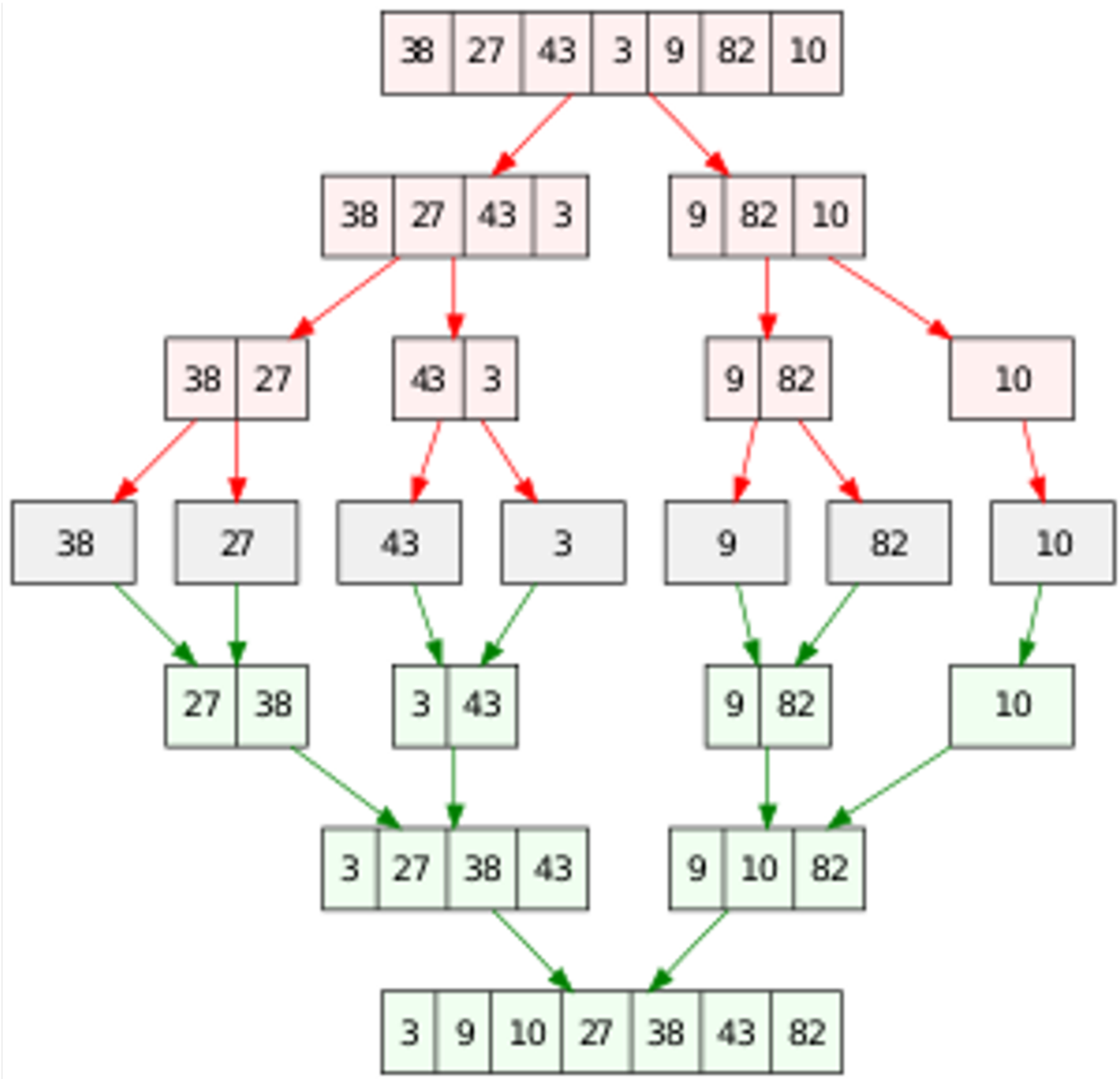
So overall runtime is:

Proof of time complexity

Claim. For large enough $c_1 > 0$, and for all $n \geq 2$, $T(n) \leq$

Proof.

Example


38 27 43 3 9 82 10
38 27 43 3
9 82 10
38 27
43 3
9 82
10
38
27
43
3
9
82
10
27 38
3 43
9 82
10
3 27 38 43
9 10 82
3 9 10 27 38 43 82

2.2 Intro to Solving Recurrences Using the Recursion Tree Method (CLRS §4.4)

Example $T(n) = 2T\left(\frac{n}{2}\right) + cn^2$

Example $T(n) = 2T\left(\frac{n}{2}\right) + c$

2.3 Quicksort (CLRS §7.1, 7.2)

Idea:

Example

```
k=PARTITION
QUICKSORT
QUICKSORT
```

2.3.1 Partition

```
pivot =
i =
for j = 1 to n-1
    if A[j]

        i =

RETURN i
```

2.3.2 Correctness of Partition (and Quicksort)

Loop invariant:

- Initialization:

- Maintenance:

- Termination:

2.3.3 Running Time of Partition and Quicksort

2.4 Integer Multiplication (Karatsuba) (KT §5.5)

Input:

Goal:

2.4.1 Elementary School Algorithm

- Time complexity of grade school addition:

- Time complexity of grade school multiplication:

2.4.2 Algorithm Using Divide & Conquer

First attempt:

Total runtime:

You try! Describe a procedure that given four integers a, b, c, d, outputs the three numbers ab, cd and $ad + bc$ and uses only **three** multiplications (four would be obvious). You are free to use as many additions and subtractions as you wish.
(Hint: Consider the product $(a + c)(b + d)$.)

Second attempt: