Week 3

Maximum Subarray Sum

Thorben Klabunde

www.th-kl.ch

06.10.2025

Agenda

1. Mini-Quiz
2. Assignment
3. Theory Recap
4. Additional Practice
5. Peer Grading

Mini-Quiz

Mini-Quiz 2

1. **CI.:** $n^8 - 800n^7 - 10000n^6 + 200n^2 \leq O(0.25n^7)$

   Answer: False, since $\lim_{n\to\infty} \frac{n^8 - 800n^7 - 10000n^6 + 200n^2}{0.25n^7} = \infty$

2. **CI.:** Let $f(n) = 4e^n$, for which of the following definitions of $g(n)$ is $f(n) \leq O(g(n))$?

True	False	
	✓	$g(n) = n^4 + n^2$
	✓	$g(n) = n^{100} + e^{100\ln(n)}$
✓		$g(n) = 2^{3n}$

Mini-Quiz 2

3. You want to show using induction that a statement $A(n)$ holds for all $n \geq 2$. Which of the following combinations of *base case* and *induction step* would form a valid proof?

✓ **a.** *Base case:* $A(2)$ holds.
Induction step: $A(k) \Rightarrow A(k+1)$ for all integers $k \geq 2$.
This is just standard induction.

✓ **b.** *Base case:* $A(2), A(3)$ holds.
Induction step: $A(k) \Rightarrow A(k+2)$ for all integers $k \geq 2$.
From base case $n = 2$ we prove it for all positive even numbers and from base case $n = 3$ we prove it for all numbers at least 3, so this covers all the numbers.

c. *Base case:* $A(2)$ holds.
Induction step: $A(k) \Rightarrow A(k+2)$ for all integers $k \geq 2$.

3. **Cl.:** Is $\sum_{i=1}^{n} i^2 \leq O(n^2 \log(n))$?

   Answer: False. Since all terms are positive, we can lower-bound the sum by its second half:

$$\sum_{i=1}^{n} i^2 \geq \sum_{i=\lfloor n/2 \rfloor}^{n} i^2 \geq \sum_{i=\lfloor n/2 \rfloor}^{n} \left(\frac{n}{2}\right)^2 \geq \left(\frac{n}{2}\right) \cdot \left(\frac{n}{2}\right)^2 = \frac{n^3}{8}$$

   The sum is in $\Omega(n^3)$, which is not in $O(n^2 \log(n))$.

Mini-Quiz 2

4. Consider the recurrence relation below, defined for $n = 2^k$, $k \in \mathbb{N}_0$:

$$T(n) = \begin{cases} 0 & \text{if } n = 1 \\ 2T(n/2) + 5n & \text{if } n > 1 \end{cases}$$

Is $T(n) \leq O(n)$?

Answer: False. This can be solved with the Master Theorem or by expansion. By expansion:

$$\begin{aligned} T(n) &= 2T(n/2) + 5n \\ &= 2(2T(n/4) + 5(n/2)) + 5n = 4T(n/4) + 2(5n) \\ &= 4(2T(n/8) + 5(n/4)) + 2(5n) = 8T(n/8) + 3(5n) \\ &\vdots \\ &= 2^k T(n/2^k) + k(5n) \end{aligned}$$

Since $n = 2^k \implies k = \log_2 n$, and the recursion stops when $n/2^k = 1$, we get:

$$T(n) = n \cdot T(1) + (\log_2 n)(5n) = n \cdot 0 + 5n \log_2 n = \Theta(n \log n)$$

Assignment

Ex. 2.2

Exercise 2.2 *Fibonacci numbers* **(1 point).**

There are a lot of neat properties of the Fibonacci numbers that can be proved by induction. Recall that the Fibonacci numbers are defined by $f_0 = 0, f_1 = 1$ and the recursion relation $f_{n+1} = f_n + f_{n-1}$ for all $n \geq 1$. For example, $f_2 = 1, f_5 = 5, f_{10} = 55, f_{15} = 610$.

Prove that $f_n \geq \frac{1}{3} \cdot 1.5^n$ for $n \geq 1$.

In your solution, you should address the base case, the induction hypothesis and the induction step.

cl. For $n \geq 1$ it holds that $f_n \geq \frac{1}{3} \cdot 1.5^n$, where f_n is the n^{th} Fibonacci number.

pr. We proceed by induction on n.

B.C. Let $n = 1$ and notice that $f_1 = 1 \geq 0.5 = \frac{1}{3} \cdot 1.5$.

Further let $n = 2$, for which: $f_2 = 1 \geq \frac{3}{4} = \frac{1}{3} \cdot \frac{9}{4} = \frac{1}{3} \cdot \left(\frac{3}{2}\right)^2$.

Ex. 2.2 - ctd.

I.H. Assume that the claim holds for some n and $n+1$ with $n \geq 1$.

Careful that you include all necessary assumptions in your I.H.

I.S. Then for $n+2$ we have:

$$
\begin{aligned}
f_{n+2} &\overset{\text{def.}}{=} f_{n+1} + f_n \\
&\overset{\text{I.H.}}{\geq} \tfrac{1}{3}\cdot(1.5)^{n+1} + \tfrac{1}{3}(1.5)^n \\
&= \tfrac{1}{3}\left(1.5^{n+1} + 1.5^n\right) \\
&= \tfrac{1}{3}\left(1.5^n\,(1.5+1)\right) \\
&= \tfrac{1}{3}\left(1.5^n \cdot 2.5\right) \\
&\geq \tfrac{1}{3}\left(1.5^n \cdot 2.25\right) \\
&= \tfrac{1}{3}\left(1.5^n \cdot 1.5^2\right) = \tfrac{1}{3}\cdot 1.5^{n+2}
\end{aligned}
$$

By the principle of math induction the claim holds for all $n \geq 1$. ∎

Ex. 2.4a)

Exercise 2.4 *Asymptotic growth of* $\sum_{i=1}^{n} \frac{1}{i}$ **(1 point).**

The goal of this exercise is to show that the sum $\sum_{i=1}^{n} \frac{1}{i}$ behaves, up to constant factors, as $\log(n)$ when n is large. Formally, we will show $\sum_{i=1}^{n} \frac{1}{i} \leq O(\log n)$ and $\log n \leq O(\sum_{i=1}^{n} \frac{1}{i})$ as functions from $\mathbb{N}_{\geq 2}$ to $\mathbb{R}^+$.

For parts (a) to (c) we assume that $n = 2^k$ is a power of 2 for $k \in \mathbb{N}_0 = \mathbb{N} \cup \{0\}$. We will generalise the result to arbitrary $n \in \mathbb{N}$ in part (d). For $j \in \mathbb{N}$, define

$$S_j = \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i}.$$

(a) For any $j \in \mathbb{N}$, prove that $S_j \leq 1$.

Hint: *Find a common upper bound for all terms in the sum and count the number of terms.*

Ex. 2.4 - ctd.

(a) cl. $\forall j \in \mathbb{N}\,(S_j \leq 1)$.

pr. Let $j \in \mathbb{N}$ be arbitrary.

$$S_j = \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i}$$

$$\leq \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{2^{j-1}}$$

$$= \underbrace{(2^{j-1}) \cdot \frac{1}{2^{j-1}}}$$

$$\leq 1$$

Step 1: Insert upper bound for variable term. Then we have a sum of the same term over and over, which we can express as a multiplication.

Step 2: Count terms.

$2^j - (2^{j-1}+1) + 1 = 2^{j-1}$ terms in the sum

Ex. 2.4b)

(b) For any $j \in \mathbb{N}$, prove that $S_j \geq \frac{1}{2}$.

(b) cl. $\forall j \in \mathbb{N} \left(S_j \geq \frac{1}{2} \right)$

pr. Let $j \in \mathbb{N}$ be arbitrary.

$$S_j = \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i}$$

$$\geq \sum_{i=2^{j-1}+1}^{2^j} \frac{1}{2^j} \quad \left| \left(\text{since } \forall j \in \mathbb{N} \left(\frac{1}{2^j} \geq 0 \wedge \frac{1}{2^{j+1}} \leq \frac{1}{2^j} \right) \right) \right.$$

see (a)

$$= \left(2^{j-1} \right) \cdot \frac{1}{2^j} \geq \frac{1}{2}$$

Ex. 2.4c)

(c) For any $k \in \mathbb{N}_0$, prove the following two inequalities

(c) cl. $\forall k \in \mathbb{N}_0 \left(\sum_{i=1}^{2^k} \frac{1}{i} \leq k+1 \;\wedge\; \sum_{i=1}^{2^k} \frac{1}{i} \geq \frac{k+1}{2} \right)$

pr. Let $k \in \mathbb{N}_0$ be arbitrary.

(i) $$\sum_{i=1}^{2^k} \frac{1}{i} = \sum_{i=1}^{1} \frac{1}{i} + \sum_{i=2^0+1}^{2^1} + \sum_{i=2^1+1}^{2^2} + \ldots + \sum_{i=2^{k-1}+1}^{2^k} = 1 + \sum_{j=1}^{k} \underbrace{\sum_{i=2^{j-1}+1}^{2^j} \frac{1}{i}}_{\leq 1 \text{ by (a)}} \overset{(5y\ a)}{\leq} 1 + k \cdot 1 = k+1$$ ☑

(ii) $$\sum_{i=1}^{2^k} \frac{1}{i} = 1 + \sum_{j=1}^{k} \underbrace{\sum_{i=2^{j-1}+1}^{2^j}}_{\geq \frac{1}{2} \text{ by (5)}} \overset{(5y\ 5)}{\geq} 1 + \frac{k}{2} = \frac{k+2}{2} \geq \frac{k+1}{2}$$ ☑

Ex. 2.4d)

(d)* For arbitrary $n \in \mathbb{N}$, prove that

$$\sum_{i=1}^{n} \frac{1}{i} \leq \log_2(n) + 2$$

and

$$\sum_{i=1}^{n} \frac{1}{i} \geq \frac{\log_2 n}{2}.$$

Hint: *Use the result from part (c) for* $k_1 = \lceil \log_2 n \rceil$ *and* $k_2 = \lfloor \log_2 n \rfloor$. *Here, for any* $x \in \mathbb{R}$, $\lceil x \rceil$ *is the smallest integer that is at least* x *and* $\lfloor x \rfloor$ *is the largest integer that is at most* x. *For example,* $\lceil 1.5 \rceil = 2$, $\lfloor 1.5 \rfloor = 1$ *and* $\lceil 3 \rceil = \lfloor 3 \rfloor = 3$. *In particular, for any* $x \in \mathbb{R}$, $x \leq \lceil x \rceil < x + 1$ *and* $x \geq \lfloor x \rfloor > x - 1$.

Ex. ~~2.3.6~~ 2.4.a)

cl. $\forall n \in \mathbb{N}\setminus\{0\}\ \left(\sum_{i=1}^{n} \frac{1}{i} \leq \log_2(n) + 2 \ \wedge \ \sum_{i=1}^{n} \frac{1}{i} \geq \frac{\log_2(n)}{2} \right)$

pr. Let $n \in \mathbb{N}\setminus\{0\}$ be arbitrary and let $k_1 = \lceil \log_2 n \rceil$ and $k_2 = \lfloor \log_2 n \rfloor$

(i) $\sum_{i=1}^{n} \frac{1}{i} \overset{(1)}{\leq} \sum_{i=1}^{2^{k_1}} \frac{1}{i} \overset{\text{by (c)}}{\leq} k_1 + 1 = \underbrace{\lceil \log_2 n \rceil}_{\leq \log_2 n + 1} + 1 \leq \log_2 n + 2$

(1) by def of k_1: $2^{k_1} \geq n$ and $\forall i \in \mathbb{N}^+ \left(\frac{1}{i} \geq 0\right)$

(ii) $\sum_{i=1}^{n} \frac{1}{i} \overset{(1)}{\geq} \sum_{i=1}^{2^{k_2}} \frac{1}{i} \geq \frac{k_2 + 1}{2} = \frac{\lfloor \log_2 n \rfloor + 1}{2} \geq \frac{(\log_2 n - 1) + 1}{2} = \frac{\log_2 n}{2}$

(1) by def of k_2: $2^{k_2} \leq n$ and $\forall i \in \mathbb{N}^+ \left(\frac{1}{i} \geq 0\right)$

Ex. 2.5a)

Exercise 2.5 *Asymptotic growth of* $\ln(n!)$.

Recall that the factorial of a positive integer n is defined as $n! = 1 \cdot 2 \cdot \cdots \cdot (n-1) \cdot n$. For the following functions n ranges over $\mathbb{N}_{\geq 2}$.

(a) Show that $\ln(n!) \leq O(n \ln n)$.

Hint: *You can use the fact that* $n! \leq n^n$ *for* $n \in \mathbb{N}_{\geq 2}$ *without proof.*

(a) Let $n \in \mathbb{N}_{\geq 2}$ be arbitrary and notice that:

$$\ln(n!) = \ln\left(\prod_{i=1}^{n} i\right) = \sum_{i=1}^{n} \ln(i) \leq \sum_{i=1}^{n} \ln(n) = n \cdot \ln(n)$$

It follows by def that $\ln(n!) \leq O(n \ln(n))$.

Ex. 2.5b)

(b) Show that $n \ln n \leq O(\ln(n!))$.

Hint: *You can use the fact that* $\left(\frac{n}{2}\right)^{\frac{n}{2}} \leq n!$ *for* $n \in \mathbb{N}_{\geq 2}$ *without proof.*

(b) Let $n \in \mathbb{N}_{\geq 2}$ be arbitrary and notice that:

$$\lim_{n\to\infty} \frac{n\ln(n)}{\ln(n!)} \overset{(1)}{\leq} \lim_{n\to\infty} \frac{n\ln(n)}{\frac{n}{2}\ln\left(\frac{n}{2}\right)} = \lim_{n\to\infty} 2 \cdot \frac{\ln(n)}{\ln\left(\frac{n}{2}\right)} = 2 \lim_{n\to\infty} \frac{\ln(n)}{\ln(n) + \ln\left(\frac{1}{2}\right)} = 2$$

$\left(\frac{\ln(n)}{\ln(n)+\ln(\frac{1}{2})} \xrightarrow{(n\to\infty)} 1\right)$

((1) by hint and since $\ln(x)$ is monotone increasing on $[2,\infty)$.)

It follows by def. that $n\ln(n) \leq O(\ln(n!))$. ∎

Theory Recap

The Maximum Subarray Sum Problem

Problem Definition

Given an array of integers arr, find the **contiguous** subarray (incl. $\varnothing$) which has the **largest sum**.

Consider the array:

$\text{arr} = [-2, 1, -3, 4, -1, 2, 1, -5, 4]$

with max. subarray sum 6, corresponding to

$[4, -1, 2, 1]$.

```
public class MaxSubarraySum {
    int[] arr;
    int n;

    MaxSubarraySum(int[] arr) {
        this.arr = arr;
        n = arr.length;
    }

    ...
}
```

We use this problem (it has practical applications as well!) to illustrate and analyze different algorithm design paradigms: brute force, divide-and-conquer, and dynamic programming.

Approach 1: Naive Cubic Time $O(N^3)$

Idea: Iterate through all possible start indices i, all possible end indices j, and for each pair (i, j), sum up all elements from i to j.

```
Result naiveSumCubic() {
    int startIdx = -1;
    int endIdx = -1;
    int maxSum = 0;
    for (int i=0; i<n; i++) {
        for (int j=i; j<n; j++) {
            int sum = 0;
            for (int k=i; k<=j; k++) {
                sum += arr[k];
            }
            if (sum > maxSum) {
                startIdx = i;
                endIdx = j;
                maxSum = sum;
            }
        }
    }
    return new Result(maxSum, startIdx, endIdx);
}
```

const.

$\in O(1)$

Analyze this loop!

$\in O(1)$

const.

How do we analyze the runtime of this program?

Break the structure down to the essentials!

$$C_1 + C_2 \cdot \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} \sum_{k=i}^{j} 1, \quad \text{for some } C_1, C_2 \in \mathbb{N}$$

$$= \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} (j - i + 1)$$

$= (\dots)$ (this is a bit of grunt work)

$= \Theta(n^3)$

Approach 2: Improved Quadratic Time $O(N^2)$

Improvement: The third loop is inefficient. We can avoid it by reusing the sum from the previous step as we extend the subarray to the right.

```
Result naiveSumSquare() {
    int startIdx = -1;
    int endIdx = -1;
    int maxSum = 0;
    for (int i=0; i<n; i++) {
        int sum = 0;
        for (int j=i; j<n; j++) {
            sum += arr[j]; // Just add the next element
            if (sum > maxSum) {
                startIdx = i;
                endIdx = j;
                maxSum = sum;
            }
        }
    }
    return new Result(maxSum, startIdx, endIdx);
}
```

Runtime:

We have eliminated one loop and obtain:

$$C_1 + C_2 \cdot \sum_{i=0}^{n-1} \sum_{j=i}^{n-1} 1$$

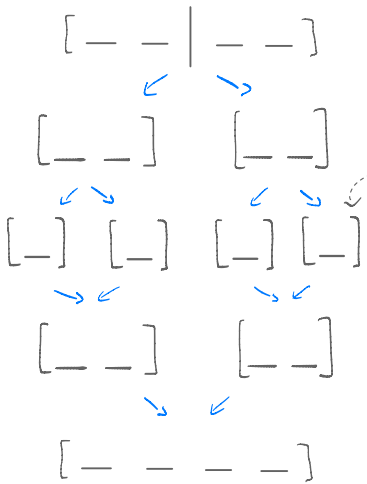
$$= C_1 + C_2 \left(\sum_{i=0}^{n-1} (n-1) - i + 1 \right)$$

$$= C_1 + C_2 \left(\underbrace{\sum_{i=0}^{n-1} n}_{} - \sum_{i=0}^{n-1} i \right)$$ ↓ Gaußsche Summenformel

$$= C_1 + C_2 \left(n^2 - \frac{n(n-1)}{2} \right)$$

$$= \Theta(n^2)$$

Approach 3a: Standard Divide and Conquer $O(N \log N)$



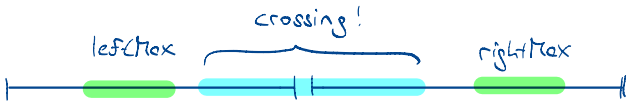
DIVIDE

Divide into smaller, more manageable problems until reaching base case with trivial solution

CONQUER

Assemble solution to larger problem from smaller problems.

Note: We have to be careful not to miss solutions!



Approach 3: Standard Divide and Conquer $O(N \log N)$

Idea: The max subarray is either (1) in the left half, (2) in the right half, or (3) it crosses the midpoint. The crossing sum is found by scanning outwards from the middle.

```
int dcNLogN(int[] arr, int left, int right) {
    // Base Case: Only one element
    if (left == right) return Math.max(0, arr[left]);

    // 1. Divide
    int mid = left + (right - left) / 2;

    // 2. Conquer
    int leftMax = dcNLogN(arr, left, mid);
    int rightMax = dcNLogN(arr, mid + 1, right);

    // 3. Combine: Find max crossing the midpoint
    int leftCrossingMax = Integer.MIN_VALUE;
    int currentSum = 0;
    for (int i = mid; i >= left; i--) {
        currentSum += arr[i];
        if (currentSum > leftCrossingMax) {
            leftCrossingMax = currentSum;
        }
    }
    int rightCrossingMax = Integer.MIN_VALUE;
    currentSum = 0;
    for (int i = mid + 1; i <= right; i++) {
        currentSum += arr[i];
        if (currentSum > rightCrossingMax) {
            rightCrossingMax = currentSum;
        }
    }
    int crossingMax = leftCrossingMax + rightCrossingMax;
    int result = Math.max(Math.max(leftMax, rightMax), crossingMax);
    return Math.max(0, result);
}
```

How do we analyze recursive algorithms?

We formulate the recurrence relation:

Notice: $T(1) = c \in \mathbb{N}$. $c \in \mathbb{N}$ & assume $n = 2^k$ for $k \in \mathbb{N}$.

Then: $T(n) = 2 \cdot T\left(\frac{n}{2}\right) + a \cdot n$

$= 2 \cdot \left(2 \cdot T\left(\frac{n}{4}\right) + a \cdot \frac{n}{2}\right) + a \cdot n$

$= 2^2 \cdot T\left(\frac{n}{2^2}\right) + (2 \cdot an)$

$= 2^2 \cdot \left(2 \cdot T\left(\frac{n}{2^3}\right) + a \cdot \frac{n}{4}\right) + (2 \cdot an)$

$= 2^3 \cdot T\left(\frac{n}{2^3}\right) + (3 \cdot an) \ldots$

In general: $2^k \cdot T\left(\frac{n}{2^k}\right) + k \cdot a \cdot n$

We divide $\log_2 n$ times until reaching the base case

$\Rightarrow T(n) = \underbrace{2^{\log_2 n} \cdot T(1)}_{c \cdot n} + \underbrace{\log_2 n \cdot a \cdot n}_{a \cdot n \cdot \log_2 n} \leq O(n \cdot \log(n))$

Approach 4: Linear DP $O(N)$

Idea: Break the problem down into subproblems that we can easily build on!

```
Result linearPassSum() {
    // DP[i] = max subarray sum ending at arr[i-1]
    int[] DP = new int[n+1];
    int maxSum = 0; int startIdx = endIdx = -1;
    int currentStartIdx = -1;
    DP[0] = 0;

    for (int i=1; i<n+1; i++) {
        int prevSum = DP[i-1];
        int currentVal = arr[i-1];

        if (prevSum + currentVal > 0) {
            DP[i] = prevSum + currentVal;
            // start index remains the same
        } else {
            DP[i] = 0; // Start a new empty subarray
            currentStartIdx = i;
        }

        if (DP[i] > maxSum) {
            maxSum = DP[i];
            endIdx = i-1;
            startIdx = currentStartIdx;
        }
    }
    return new Result(maxSum, startIdx, endIdx);
}
```

← empty sum!

Correctness:

Meaning of each entry (DP[i]):

DP[i] = Max. subarray sum ending at arr[i-1]

Recursion:

DP[i] = DP[i-1] + arr[i-1] if >0 else Math.max(0, arr[i-1])

Notice that the maximum subarray sum must append to the maximum subarray sum ending one index before.

Correctness:

Every max. subarray sum ends at an index.

Since we compute each entry correctly, we compute the max. subarray sum correctly.

Approach 4: Linear DP $O(N)$

Idea: Break the problem down into subproblems that we can easily build on!

```
Result linearPassSum() {
    // DP[i] = max subarray sum ending at arr[i-1]
    int[] DP = new int[n+1];
    int maxSum = 0; int startIdx = endIdx = -1;
    int currentStartIdx = -1;
    DP[0] = 0;

    for (int i=1; i<n+1; i++) {
        int prevSum = DP[i-1];
        int currentVal = arr[i-1];

        if (prevSum + currentVal > 0) {
            DP[i] = prevSum + currentVal;
            // start index remains the same
        } else {
            DP[i] = 0; // Start a new empty subarray
            currentStartIdx = i;
        }

        if (DP[i] > maxSum) {
            maxSum = DP[i];
            endIdx = i-1;
            startIdx = currentStartIdx;
        }
    }
    return new Result(maxSum, startIdx, endIdx);
}
```

Runtime:

B.C. $O(1)$

\+

Rec. $O(n)$ entries $\times$ $O(1)$ / entry

\+

Extract Sol. $O(1)$

$O(n)$

Divide-and-Conquer - Geht es besser?

Recall: The max sum is the maximum of: (1) max sum in left half, (2) max sum in right half, and (3) max sum crossing the midpoint.

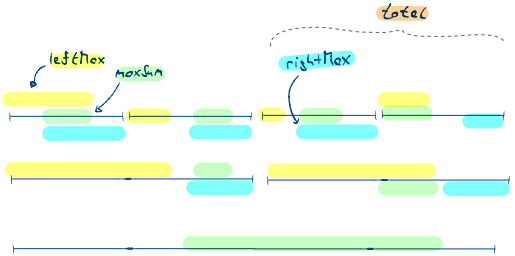
```
DCResult divideAndConquerSum(int left, int right) {
    if (right == left) { // Base Case
        int val = Math.max(0, arr[left]);
        int idx = (arr[left] >= 0) ? left : -1;
        return new DCResult(val, idx, val, idx, val, idx, idx, arr[left]);
    }
    int mid = left+((right-left)/2);
    DCResult l = divideAndConquerSum(left, mid);
    DCResult r = divideAndConquerSum(mid+1, right);

    // Combine step (O(1) work)
    int leftMax = Math.max(l.leftMax, l.total + r.leftMax);
    int rightMax = Math.max(r.rightMax, r.total + l.rightMax);
    int maxSum;
    if (l.rightMax + r.leftMax > Math.max(l.maxSum, r.maxSum)) {
        maxSum = l.rightMax + r.leftMax;
    } else {
        maxSum = Math.max(l.maxSum, r.maxSum);
    }
    int total = l.total + r.total;
    // (index tracking logic omitted for brevity)
    return new DCResult(leftMax, ..., rightMax, ..., maxSum, ..., total);
}
```

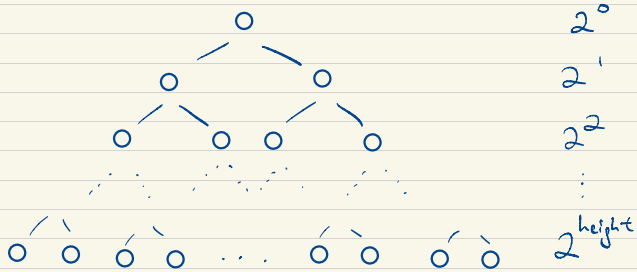
Runtime:

All the required information is there.

No need to recompute!



Visualizing Runtime of Recursive DnC



2^0
2^1
2^2
$\vdots$
2^{height}

$$\sum_{i=1}^{depth} 2^i = 2^{i+1} - 1 \text{ nodes}$$

$\Rightarrow$ For height of $\log_2 n$ we get $2^{\log_2 n + 1} - 1 = 2n - 1$ nodes, each computable in $O(1)$

Thus, overall $O(n)$.

Additional Practice

Counting Loop Iterations

/ 4 P

a) *Counting loop iterations:* For the following code snippets, derive an expression for the number of times f is called. Simplify the expression as much as possible and state it in Θ-notation.

i) Snippet 1:

Algorithm 1

```
for j = 1, ..., n do
    for k = j^2, ..., (j+1)^2 do
        f()
```

(a) Since f is called once in each iteration of the inner for-loop, the number of calls amounts to:

$$\sum_{j=1}^{n} \sum_{k=j^2}^{(j+1)^2} 1 = \sum_{j=1}^{n} (j+1)^2 - j^2 + 1 = \sum_{j=1}^{n} 2j + 2 = 2 \cdot \sum_{j=1}^{n} j + \sum_{j=1}^{n} 2 = 2 \cdot \frac{n(n+1)}{2} + 2n$$

$$= n^2 + 3n = \Theta(n^2)$$

Peer Grading

Peer-Grading Exercise

This week's peer-grading exercise is **Exercise 2.4**

Each group grades the group below in the table I sent you (resp. the last one grades the first one). Please send the other group your solution. If you don't get their solution, please contact me so I can send it to you.

Questions?

Bounding Sums with Integrals

Recall From Last Week

We showed that for any constant $k \geq 1$, the sum of the first n k-th powers is in $\Theta(n^{k+1})$.

$$\sum_{i=1}^{n} i^k = \Theta(n^{k+1})$$

We proved this by finding constants c_1 and c_2 to establish the upper and lower bounds directly from the sum's properties.

A New Technique: Using Integrals

The core idea: The sum $\sum_{i=1}^{n} f(i)$ can be seen as the total area of n rectangles, each with width 1 and height $f(i)$. This area can be bounded by the area under the curve of the continuous function $f(x)$.

Bounding Sums with Integrals

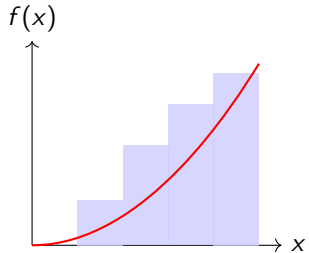
For any **monotonically increasing** function $f(x)$:

Lower Bound: The sum is at least the area under the curve from 0 to n.

$$\sum_{i=1}^{n} f(i) \geq \int_{0}^{n} f(x)\,dx$$

Upper Bound: The sum is at most the area under the curve from 1 to $n+1$.

$$\sum_{i=1}^{n} f(i) \leq \int_{1}^{n+1} f(x)\,dx$$



Area under curve bounds sum

Bounding Sums with Integrals

Example: Applying this to $\sum_{i=1}^{n} i^k$

Let $f(x) = x^k$. This function is monotonically increasing for $x \geq 0$.

Lower Bound (Ω):

$$\sum_{i=1}^{n} i^k \geq \int_0^n x^k \, dx = \left[\frac{x^{k+1}}{k+1}\right]_0^n = \frac{n^{k+1}}{k+1} \implies \Omega(n^{k+1})$$

Upper Bound (O):

$$\sum_{i=1}^{n} i^k \leq \int_1^{n+1} x^k \, dx = \left[\frac{x^{k+1}}{k+1}\right]_1^{n+1} = \frac{(n+1)^{k+1}}{k+1} - \frac{1}{k+1} \implies O(n^{k+1})$$

Since the sum is both $\Omega(n^{k+1})$ and $O(n^{k+1})$, we conclude it is $\Theta(n^{k+1})$.

Note on Monotonically Decreasing Functions

What if the function is decreasing?

The same technique works, but the inequalities for the bounds are flipped. For a monotonically **decreasing** function $f(x)$:

$$\int_1^{n+1} f(x)\,dx \leq \sum_{i=1}^{n} f(i) \leq f(1) + \int_1^n f(x)\,dx$$

Note: The upper bound is the first term $f(1)$ plus the integral over the rest of the terms.

Your Turn!

Claim: The Harmonic Series $H_n = \sum_{i=1}^{n} \frac{1}{i} \in \Theta(\log n)$.

Your Turn!

Example: The **Harmonic Series** $H_n = \sum_{i=1}^{n} \frac{1}{i}$

Let $f(x) = 1/x$ and notice that $f(x)$ is **monotonically decreasing** for $x > 0$. Accordingly:

Lower Bound (Ω):

$$\sum_{i=1}^{n} \frac{1}{i} \geq \int_{1}^{n+1} \frac{1}{x}\, dx = \big[\ln x\big]_{1}^{n+1} = \ln(n+1) - \ln(1) = \ln(n+1) \geq \Omega(\log n)$$

Upper Bound (O):

$$\sum_{i=1}^{n} \frac{1}{i} \leq f(1) + \int_{1}^{n} \frac{1}{x}\, dx = 1 + \big[\ln x\big]_{1}^{n} = 1 + \ln(n) - \ln(1) = 1 + \ln(n) \leq O(\log n)$$

Therefore, the Harmonic series grows logarithmically: $H_n = \Theta(\log n)$.