

# Divide and Conquer Optimization

---

Arnar Bjarni Arnarson

**Árangursrík forritun og lausn verkefna**

School of Computer Science

Reykjavík University

# Divide and Conquer Optimization

---

- The problem used here as an example requires that you know Prefix Sums as an orthogonal part of the problem.

# Preface

- The problem used here as an example requires that you know Prefix Sums as an orthogonal part of the problem.
- The idea is to compute all prefix sums of the array.

# Preface

- The problem used here as an example requires that you know Prefix Sums as an orthogonal part of the problem.
- The idea is to compute all prefix sums of the array.
- Suppose you have an array of integers  $A_0, A_1, \dots, A_{N-1}$ .

# Preface

- The problem used here as an example requires that you know Prefix Sums as an orthogonal part of the problem.
- The idea is to compute all prefix sums of the array.
- Suppose you have an array of integers  $A_0, A_1, \dots, A_{N-1}$ .
- Let  $S(i, j) = A_i + A_{i+1} + \dots + A_{j-1} + A_j$ .

# Preface

- The problem used here as an example requires that you know Prefix Sums as an orthogonal part of the problem.
- The idea is to compute all prefix sums of the array.
- Suppose you have an array of integers  $A_0, A_1, \dots, A_{N-1}$ .
- Let  $S(i, j) = A_i + A_{i+1} + \dots + A_{j-1} + A_j$ .
- Compute and store all values of  $S(0, j)$  for all  $j$  such that  $0 \leq j < N$ .

# Preface

- The problem used here as an example requires that you know Prefix Sums as an orthogonal part of the problem.
- The idea is to compute all prefix sums of the array.
- Suppose you have an array of integers  $A_0, A_1, \dots, A_{N-1}$ .
- Let  $S(i, j) = A_i + A_{i+1} + \dots + A_{j-1} + A_j$ .
- Compute and store all values of  $S(0, j)$  for all  $j$  such that  $0 \leq j < N$ .
- Now you can compute  $S(i, j) = S(0, j) - S(0, i - 1)$  in constant time for any  $i$  and  $j$ .



# Guards

- There are  $1 \leq N \leq 8000$  prisoners, each in their own cell, and  $1 \leq G \leq 3000$  guards.

# Guards

- There are  $1 \leq N \leq 8000$  prisoners, each in their own cell, and  $1 \leq G \leq 3000$  guards.
- Each prisoner  $i$  has an intelligence  $1 \leq S_i \leq 10^9$ .

# Guards

- There are  $1 \leq N \leq 8000$  prisoners, each in their own cell, and  $1 \leq G \leq 3000$  guards.
- Each prisoner  $i$  has an intelligence  $1 \leq S_i \leq 10^9$ .
- Each jail cell is assigned to exactly one guard.

# Guards

- There are  $1 \leq N \leq 8000$  prisoners, each in their own cell, and  $1 \leq G \leq 3000$  guards.
- Each prisoner  $i$  has an intelligence  $1 \leq S_i \leq 10^9$ .
- Each jail cell is assigned to exactly one guard.
- Each guard can only watch over a contiguous range of prisoners.

# Guards

- There are  $1 \leq N \leq 8000$  prisoners, each in their own cell, and  $1 \leq G \leq 3000$  guards.
- Each prisoner  $i$  has an intelligence  $1 \leq S_i \leq 10^9$ .
- Each jail cell is assigned to exactly one guard.
- Each guard can only watch over a contiguous range of prisoners.
- If the guard watching prisoner  $i$  is watching over  $k$  cells, then the prisoner's escaping potential is  $kS_i$ .

# Guards

- There are  $1 \leq N \leq 8000$  prisoners, each in their own cell, and  $1 \leq G \leq 3000$  guards.
- Each prisoner  $i$  has an intelligence  $1 \leq S_i \leq 10^9$ .
- Each jail cell is assigned to exactly one guard.
- Each guard can only watch over a contiguous range of prisoners.
- If the guard watching prisoner  $i$  is watching over  $k$  cells, then the prisoner's escaping potential is  $kS_i$ .
- Your goal is to assign the cells to guards in a way that minimizes the total escaping potential over all prisoners.

## Constructing a solution

- Let  $\text{dp}(j, k)$  denote the optimal answer up to the  $j$ th prisoner with  $k$  guards.

## Constructing a solution

- Let  $\text{dp}(j, k)$  denote the optimal answer up to the  $j$ th prisoner with  $k$  guards.
- Let  $C(i, j) = (j - i + 1) \cdot \sum_{k=i}^j S_k$ .



## Constructing a solution

- Let  $\text{dp}(j, k)$  denote the optimal answer up to the  $j$ th prisoner with  $k$  guards.
- Let  $C(i, j) = (j - i + 1) \cdot \sum_{k=i}^j S_k$ .
- Then we have  $\text{dp}(i, 1) = C(0, i)$  as a base case.

## Constructing a solution

- Let  $\text{dp}(j, k)$  denote the optimal answer up to the  $j$ th prisoner with  $k$  guards.
- Let  $C(i, j) = (j - i + 1) \cdot \sum_{k=i}^j S_k$ .
- Then we have  $\text{dp}(i, 1) = C(0, i)$  as a base case.
- When assigning a new guard, let's fix the end index of the segment. We must then find the optimal starting index.

## Constructing a solution

- Let  $\text{dp}(j, k)$  denote the optimal answer up to the  $j$ th prisoner with  $k$  guards.
- Let  $C(i, j) = (j - i + 1) \cdot \sum_{k=i}^j S_k$ .
- Then we have  $\text{dp}(i, 1) = C(0, i)$  as a base case.
- When assigning a new guard, let's fix the end index of the segment. We must then find the optimal starting index.
- We try all start indices

$$\text{dp}(j, k) = \min_{0 \leq i < j} (\text{dp}(i, k - 1) + C(i + 1, j))$$

## Constructing a solution

- Let  $\text{dp}(j, k)$  denote the optimal answer up to the  $j$ th prisoner with  $k$  guards.
- Let  $C(i, j) = (j - i + 1) \cdot \sum_{k=i}^j S_k$ .
- Then we have  $\text{dp}(i, 1) = C(0, i)$  as a base case.
- When assigning a new guard, let's fix the end index of the segment. We must then find the optimal starting index.
- We try all start indices

$$\text{dp}(j, k) = \min_{0 \leq i < j} (\text{dp}(i, k - 1) + C(i + 1, j))$$

- Our state space is  $\mathcal{O}(NG)$  and each state can be computed in  $\mathcal{O}(N)$  time.

## Constructing a solution

- Let  $\text{dp}(j, k)$  denote the optimal answer up to the  $j$ th prisoner with  $k$  guards.
- Let  $C(i, j) = (j - i + 1) \cdot \sum_{k=i}^j S_k$ .
- Then we have  $\text{dp}(i, 1) = C(0, i)$  as a base case.
- When assigning a new guard, let's fix the end index of the segment. We must then find the optimal starting index.
- We try all start indices

$$\text{dp}(j, k) = \min_{0 \leq i < j} (\text{dp}(i, k - 1) + C(i + 1, j))$$

- Our state space is  $\mathcal{O}(NG)$  and each state can be computed in  $\mathcal{O}(N)$  time.
- Time complexity is  $\mathcal{O}(N^2G)$ , which is too slow.

# Implementation - Initial Definitions

```
#include <bits/stdc++.h>
using namespace std;
typedef long long ll;
const ll INF = 80'000'000'000'000'000LL;

ll arr[8000];
ll prefix_sum[8001];
ll mem[3001][8001];
ll range_sum(int left, int right) {
    return prefix_sum[right] - prefix_sum[left-1];
}

ll cost(ll left, ll right) {
    return range_sum(left, right) * (right - left + 1LL);
}
```

# Naive Implementation - Computing Each Layer

```
void compute(int level, int n) {  
    for (int end = 0; end < n; end++) {  
        ll tmp = INF;  
        for (int start = 0; start <= end; start++) {  
            tmp = min(tmp,  
                      (start ? mem[level - 1][start - 1] : 0)  
                      + cost(start, end));  
        }  
        mem[level][end] = tmp;  
    }  
}
```

# Naive Implementation - Main

```
int main()
{
    int n, g;
    cin >> n >> g;
    prefix_sum[0] = 0;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        prefix_sum[i+1] = prefix_sum[i] + arr[i];
    }
    for (int i = 0; i < n; i++) {
        mem[0][i] = cost(0, i);
    }
    for (int guards = 2; guards <= g; guards++) {
        compute(guards - 1, n);
    }
    cout << mem[g - 1][n - 1] << endl;
    return 0;
}
```



# Optimizing

- Let  $\text{opt}(j, k)$  be the optimal splitting point, or the value of  $i$  which minimizes the previous expression.

# Optimizing

- Let  $\text{opt}(j, k)$  be the optimal splitting point, or the value of  $i$  which minimizes the previous expression.
- We note that  $\text{opt}(j - 1, k) \leq \text{opt}(j, k) \leq \text{opt}(j + 1, k)$ .

# Optimizing

- Let  $\text{opt}(j, k)$  be the optimal splitting point, or the value of  $i$  which minimizes the previous expression.
- We note that  $\text{opt}(j - 1, k) \leq \text{opt}(j, k) \leq \text{opt}(j + 1, k)$ .
- This allows us to divide and conquer.

# Optimizing

- Let  $\text{opt}(j, k)$  be the optimal splitting point, or the value of  $i$  which minimizes the previous expression.
- We note that  $\text{opt}(j - 1, k) \leq \text{opt}(j, k) \leq \text{opt}(j + 1, k)$ .
- This allows us to divide and conquer.
- First compute  $\text{dp}(N/2, k)$  and note the value of  $\text{opt}(N/2, k)$ .

# Optimizing

- Let  $\text{opt}(j, k)$  be the optimal splitting point, or the value of  $i$  which minimizes the previous expression.
- We note that  $\text{opt}(j - 1, k) \leq \text{opt}(j, k) \leq \text{opt}(j + 1, k)$ .
- This allows us to divide and conquer.
- First compute  $\text{dp}(N/2, k)$  and note the value of  $\text{opt}(N/2, k)$ .
- With that value in mind, compute  $\text{dp}(N/4, k)$  and  $\text{dp}(3N/4, k)$ .

# Optimizing

- Let  $\text{opt}(j, k)$  be the optimal splitting point, or the value of  $i$  which minimizes the previous expression.
- We note that  $\text{opt}(j - 1, k) \leq \text{opt}(j, k) \leq \text{opt}(j + 1, k)$ .
- This allows us to divide and conquer.
- First compute  $\text{dp}(N/2, k)$  and note the value of  $\text{opt}(N/2, k)$ .
- With that value in mind, compute  $\text{dp}(N/4, k)$  and  $\text{dp}(3N/4, k)$ .
- Repeat this process, computing the left and right side, tracking the minimum and maximum possible value of  $\text{opt}(j, k)$ .

## Analyzing the complexity

- Let  $l$  and  $r$  be the values we're tracking such that  $l \leq \text{opt}(j, k) \leq r$ .

## Analyzing the complexity

- Let  $l$  and  $r$  be the values we're tracking such that  $l \leq \text{opt}(j, k) \leq r$ .
- At each step we iterate from  $l$  to  $r$ . Doesn't look like we've made improvements...



## Analyzing the complexity

- Let  $l$  and  $r$  be the values we're tracking such that  $l \leq \text{opt}(j, k) \leq r$ .
- At each step we iterate from  $l$  to  $r$ . Doesn't look like we've made improvements...
- Note the level of the recursion, so level 1 is where we compute  $j = N/2$ , level 2 is where we compute  $j = N/4$  and  $j = 3N/4$ , and so on.

## Analyzing the complexity

- Let  $l$  and  $r$  be the values we're tracking such that  $l \leq \text{opt}(j, k) \leq r$ .
- At each step we iterate from  $l$  to  $r$ . Doesn't look like we've made improvements...
- Note the level of the recursion, so level 1 is where we compute  $j = N/2$ , level 2 is where we compute  $j = N/4$  and  $j = 3N/4$ , and so on.
- There are at most  $\mathcal{O}(\log N)$  levels.

## Analyzing the complexity

- Let  $l$  and  $r$  be the values we're tracking such that  $l \leq \text{opt}(j, k) \leq r$ .
- At each step we iterate from  $l$  to  $r$ . Doesn't look like we've made improvements...
- Note the level of the recursion, so level 1 is where we compute  $j = N/2$ , level 2 is where we compute  $j = N/4$  and  $j = 3N/4$ , and so on.
- There are at most  $\mathcal{O}(\log N)$  levels.
- At each level we will do  $\mathcal{O}(N)$  work, since there is no overlap for values of  $j$  at the same level.

## Analyzing the complexity

- Let  $l$  and  $r$  be the values we're tracking such that  $l \leq \text{opt}(j, k) \leq r$ .
- At each step we iterate from  $l$  to  $r$ . Doesn't look like we've made improvements...
- Note the level of the recursion, so level 1 is where we compute  $j = N/2$ , level 2 is where we compute  $j = N/4$  and  $j = 3N/4$ , and so on.
- There are at most  $\mathcal{O}(\log N)$  levels.
- At each level we will do  $\mathcal{O}(N)$  work, since there is no overlap for values of  $j$  at the same level.
- Note it does not matter how balanced  $\text{opt}(j, k)$  is, we always do linear work at a level.

## Analyzing the complexity

- Let  $l$  and  $r$  be the values we're tracking such that  $l \leq \text{opt}(j, k) \leq r$ .
- At each step we iterate from  $l$  to  $r$ . Doesn't look like we've made improvements...
- Note the level of the recursion, so level 1 is where we compute  $j = N/2$ , level 2 is where we compute  $j = N/4$  and  $j = 3N/4$ , and so on.
- There are at most  $\mathcal{O}(\log N)$  levels.
- At each level we will do  $\mathcal{O}(N)$  work, since there is no overlap for values of  $j$  at the same level.
- Note it does not matter how balanced  $\text{opt}(j, k)$  is, we always do linear work at a level.
- Time complexity is now  $\mathcal{O}(NG \log N)$ , so fast enough.

# Optimized Implementation - Computing Each Layer

```
void compute(int level, int l, int r, int optl, int optr) {
    if (l > r) return;
    int mid = (l+r)/2;
    pair<ll, int> best = {INF, -1};
    for (int k = optl; k <= min(mid, optr); k++) {
        best = min(best,
            {(k ? mem[level - 1][k - 1] : 0LL) + cost(k, mid), k});
    }
    mem[level][mid] = best.first;
    int opt = best.second;
    compute(level, l, mid-1, optl, opt);
    compute(level, mid+1, r, opt, optr);
}
```

# Optimized Implementation - Main

```
int main()
{
    int n, g;
    cin >> n >> g;
    prefix_sum[0] = 0;
    for (int i = 0; i < n; i++) {
        cin >> arr[i];
        prefix_sum[i+1] = prefix_sum[i] + arr[i];
    }
    for (int i = 0; i < n; i++) {
        mem[0][i] = cost(0, i);
    }
    for (int guards = 2; guards <= g; guards++) {
        compute(guards-1, 0, n-1, 0, n-1);
    }
    cout << mem[g-1][n-1] << endl;
    return 0;
}
```

## When can we use this?

- When are the optimal splitting points are monotonic?



## When can we use this?

- When are the optimal splitting points are monotonic?
- When the cost function  $C$  satisfies the quadrangle inequality (but other times too)!

## When can we use this?

- When are the optimal splitting points are monotonic?
- When the cost function  $C$  satisfies the quadrangle inequality (but other times too)!
- When  $C(a, c) + C(b, d) \leq C(a, d) + C(b, c)$  for all  $a < b < c < d$ .

## When can we use this?

- When are the optimal splitting points are monotonic?
- When the cost function  $C$  satisfies the quadrangle inequality (but other times too)!
- When  $C(a, c) + C(b, d) \leq C(a, d) + C(b, c)$  for all  $a < b < c < d$ .
- Intuitively, this means longer segments are worse than shorter segments.

## When can we use this?

- When are the optimal splitting points are monotonic?
- When the cost function  $C$  satisfies the quadrangle inequality (but other times too)!
- When  $C(a, c) + C(b, d) \leq C(a, d) + C(b, c)$  for all  $a < b < c < d$ .
- Intuitively, this means longer segments are worse than shorter segments.
- It is usually not that difficult to prove the quadrangle inequality holds when it does.

## When can we use this?

- When are the optimal splitting points are monotonic?
- When the cost function  $C$  satisfies the quadrangle inequality (but other times too)!
- When  $C(a, c) + C(b, d) \leq C(a, d) + C(b, c)$  for all  $a < b < c < d$ .
- Intuitively, this means longer segments are worse than shorter segments.
- It is usually not that difficult to prove the quadrangle inequality holds when it does.
- Once you've proven it for a DP pattern like shown before, you know you can use this method.

## When can we use this?

- When are the optimal splitting points are monotonic?
- When the cost function  $C$  satisfies the quadrangle inequality (but other times too)!
- When  $C(a, c) + C(b, d) \leq C(a, d) + C(b, c)$  for all  $a < b < c < d$ .
- Intuitively, this means longer segments are worse than shorter segments.
- It is usually not that difficult to prove the quadrangle inequality holds when it does.
- Once you've proven it for a DP pattern like shown before, you know you can use this method.
- The Convex Hull Trick can often be used in the same tasks to which this method applies.

## Try on these problems!

- Guards
- Split the Sequences
- Partition Game
- The Bakery