Sqrt Decomposition

CS 491 – Competitive Programming

Dr. Mattox Beckman

University of Illinois at Urbana-Champaign
Department of Computer Science

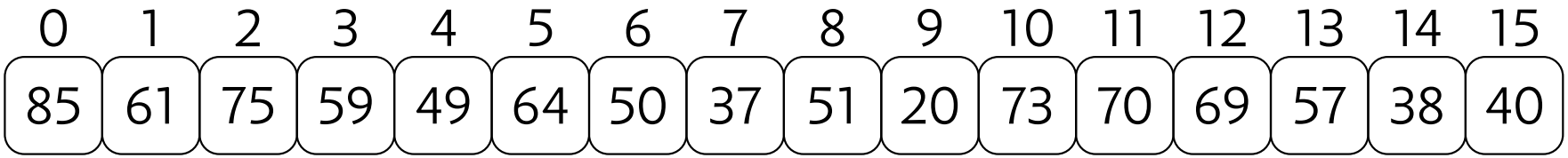
Fall 2024

Objectives

- Use sqrt decomposition to improve the time complexity of large problems.

Running Example

- Consider the following array:



- What is an algorithm, given i and j, of returning the sum between these numbers (inclusive)?
- What is an algorithm, given i and j, of returning the max between these numbers (inclusive)?

Code for Sum

```
1  vi run;
2  int a = 0;
3
4  run.push_back(0); // sentinel
5  for(int i: data) {
6   a += data;
7   run.push_back(a);
8  }
9
10 int sum(int i, int j) {
11    return run[j+1] - run[i];
12 }
```

- We can't do a "running max" though.

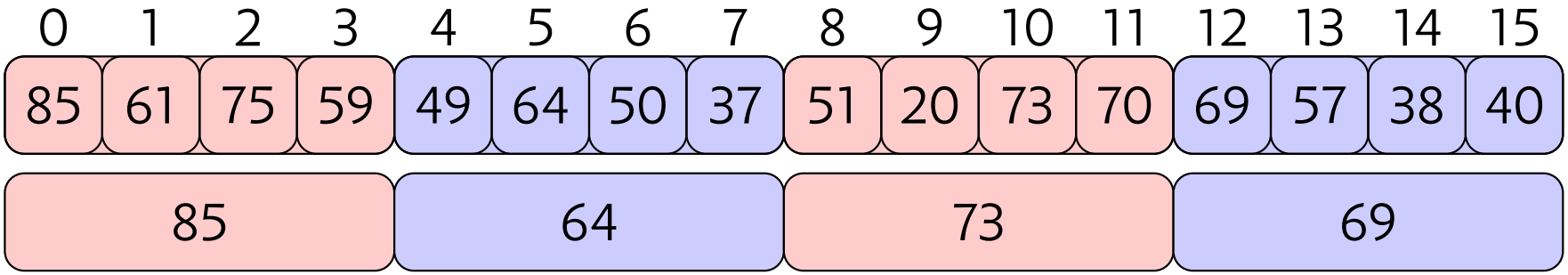
Kotlin Version

```kotlin
val size = readln().toInt()
val data = readln().split(' ').map { it.toInt() }
val run = data.runningFold(0) { acc, num ->
    acc + num }.drop(1)

fun sum(run : List<Int>, i : Int, j : Int) : Int {
   return run[j+1] - run[i]
}
```

Solution

- We can create a separate array to handle each block of $\sqrt{n}$ size.



- What is the max number between...
  - 0 and 2?
  - 4 and 11?
  - 3 and 8?
- What is the resulting time complexity?

Preprocessing Code

- sq contains the sqrt decompositions.
- data contains the raw data.

```
1  vi data, sq;
2  int n,a,d;
3
4  cin >> n;
5  int s = sqrt(n);
6  sq = vi(s+1);
7
8  for(int i=0,j=0; i < s && j < n; ++i ) {
9    cin >> d; data.push_back(d);
10   sq[i] = d;
11   for(k=1; k<s; ++j, ++k) {
12      cin >> d; data.push_back(d);
13      sq[i] = max(sq[i],d);
14    }
15 }
```

Preprocessing Code, Kotlin

```kotlin
import kotlin.math.sqrt

val size = readln().toInt()
val data : List<Int> = readln().split(' ').map {
    it.toInt() }

val nsq = sqrt(data.size.toDouble()).toInt()
val chunk = data.chunked(nsq)
val sq = chunk.map { sublist -> sublist.max() }
```

Search

Note this ignores some edge cases

```
1  int findMax(int i, int j, vi &data, vi &sq, int s) {
2    int a;
3    a = data[i];
4    while (i % s > 0 && i <= j) {
5      a = max(a,data[i]);
6      ++i;
7    }
8
9    while ( (j+1) % s > 0) {
10     a = max(a,data[j]);
11     --j;
12   }
13
14   for(k = i/s; k<=j/s; ++k)
15     a = max(a,sq[k]);
16 }
```

Kotlin Version

```
1  fun findMax(i: Int , j : Int, data : List<Int>, sq : List<I
2      var a = data[i]
3      val iup = minOf(j, i + s - (i % s) - 1)
4      val jdown = j - (j % s)
5      val b1 = i / s
6      val b2 = j / s
7
8      if (b1 == b2) { // Same block
9          return maxOf(a,data.slice(i..j).max())}
10     if (i % s > 0) {a = maxOf(a,data.slice(i..iup).max())}
11     if (jdown <= j) {a = maxOf(a,data.slice(jdown..j).max())
12     if (b1+1 < b2) {a = maxOf(a,sq.slice(b1+1..b2-1).max())}
13     return a
14 }
15
16
```