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:

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
85	61	75	59	49	64	50	37	51	20	73	70	69	57	38	40

- 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

```
vi run;
int a = 0;

run.push_back(0); // sentinel
for(int i: data) {
 a += data;
 run.push_back(a);
}

int sum(int i, int j) {
   return run[j+1] - run[i];
}
```

- We can’t do a “running max” though.

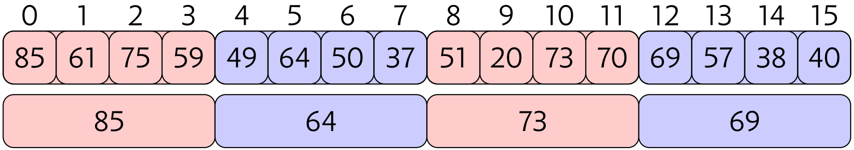
Kotlin Version

```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.

```
vi data, sq;
int n,a,d;

cin >> n;
int s = sqrt(n);
sq = vi(s+1);

for(int i=0,j=0; i < s && j < n; ++i ) {
  cin >> d; data.push_back(d);
  sq[i] = d;
  for(k=1; k<s; ++j, ++k) {
     cin >> d; data.push_back(d);
     sq[i] = max(sq[i],d);
   }
}
```

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

```
int findMax(int i, int j, vi &data, vi &sq, int s) {
  int a;
  a = data[i];
  while (i % s > 0 && i <= j) {
    a = max(a,data[i]);
    ++i;
  }

  while ( (j+1) % s > 0) {
    a = max(a,data[j]);
    --j;
  }

  for(k = i/s; k<=j/s; ++k)
    a = max(a,sq[k]);
}
```