RAM:

* Assignments
* Retrieve values
* Computations
* Return

Maximum value in array of integers

INPUT: A - array of size n



OUTPUT: integer

arrayMax(A) {



currentMax <- A[0] **2**



For(I <- 1 to u) 1 + 2(n – 1)



If(A[i] > currentMax)



currentMax <-A[i]



return currentMax;



}

INPUT: A – array, n – integer

OUTPUT: integer

recursiveMax(A, n) {

if (n == 1) **1**

return A[0] **2**

return max(recrusiveMax(A, n – 1), A[n – 1]) **6**

**T(n) = {3, T(n – 1) + 7}**

**3 + 7(n – 1) =>**

**7n - 4**

Big-o: Complexity

* Log(n), log2(n), √n, n, nlog(n), n2, n3, 2n
* Much simpler

F(n) is O(g(n)) if f(n) is asymptotically less than or equal g(n)

Prove:

* Let f(n) and g(n) mapping non-negative numbers to real numbers
* We say that f(n) is O(g(n))
* c > 0
* n0 > 1
* f(n) ≤ c \* g(n), n ≥ no
* 7n – 4 is O (g(n))
* 7n – 4 ≤ n + 4, n ≥ 1

Rules:

1. if d(n) is O(f(n)), then a\*d(n) is O(f(n)), for any constant d > 0
2. if d(n) is O(f(n) and e(n) is O(g(n)) then d(n) + e(n) is O(f(n) + g(n))
3. “ “ “ “ “ “ “ “ “ “ d(n) \* e(n) is O(f(n) \* g(n))
4. “ “ “ “ “ “ f(n) is O(g(n)) “ d(n) is O(g(n))
5. “ f(n) is a polynomial of degree d then f(n) is O(nd)
6. nx is O(dn) for any x > 0 and d > 1
7. Logxn is O(nx) for any x>0 and y > 0

Example:

* 2n3 + 4n2logn
  + rule 1: drop 4 and 2
* O(n3) + n2logn
  + rule 7: log1n = o(n)
* O(n3) + O(n2) \* O(n)
  + rule 3: multiply n
* O(n3) + O(n3)
  + rule 2: add
* O(2n3)
  + rule 1: drop constant
* O(n3)

Practice:

* [-2, -4, 3, -1, 5, 6, -7, -2, 4, -3, 2]
* Find the connected subset (of any length) with the maximum sum
* Best solution is O(n)