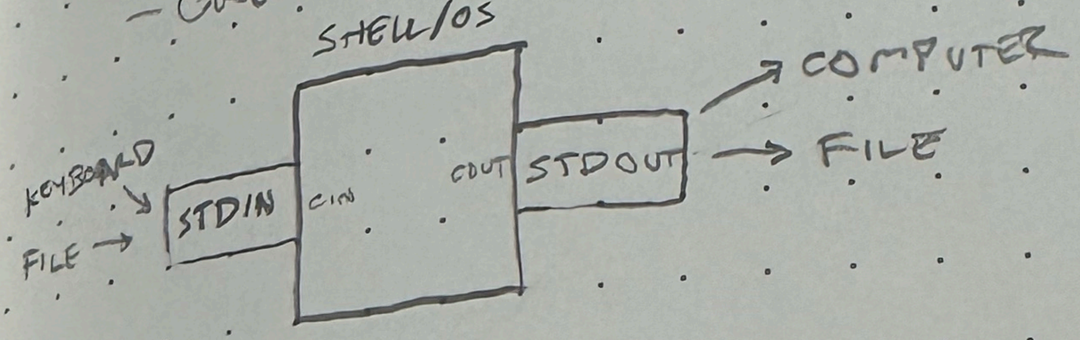
DATA STRUCTURES [02/07/24]

TESTING FOR ASSIGNMENT #1
- BOUNDARIES (MOST AND LEAST)

TO TEST WITH MILLIONS OF LINES:
- FIND ON INTERNET
- CREATE A PROGRAM



(REDIRECT VIA COMMAND LINE)

CAT COMMAND DISPLAYS FILE
TAC DISPLAYS FILE IN REVERSE
'tac TEST.TXT > TESTREVERSED.TXT'
diff COMMAND GIVES DIFFERENCE BTWN 2 FILES
'diff myout.txt BIG.TEST.TXT'

REMOVE TESTS PRIOR TO TURNING IN

USE "time" COMMAND

NO CUT/PAST WITH PROGRAM INPUT!

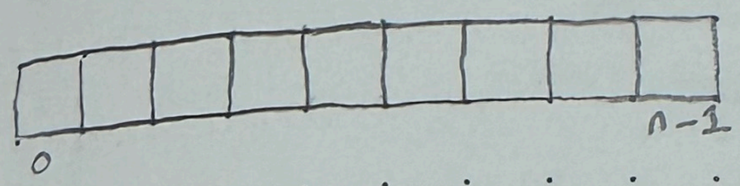
./a.out > MYOUTPUT.TXT
./a.out < MY INPUT.TXT
./ < IN >OUT

DSA [02/07/24]

ALGORITHM (PROGRAM) COMPLEXITY
1. TIME (RELATIVE ACCORDING TO A METRIC)
2. SPACE (MEMORY USAGE)

SPACE COMPLEXITY - HOW MUCH MEMORY DO WE NEED TO RUN THE ALGORITHM

EX. AN ARRAY LIKE "INPUT" FOR AN ALG. (SIZE n)



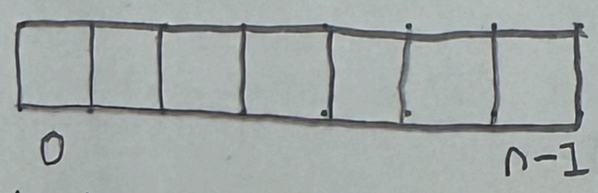
FOR FINDING SOMETHING LIKE A MAX, IT TAKES A CONSTANT AMOUNT OF STORAGE ("IN-PLACE" ALG.)

"BIG O"

LINEAR ($O(n)$) - NOT IN PLACE. NEEDS MORE MEMORY
- SOME FACTOR OF n

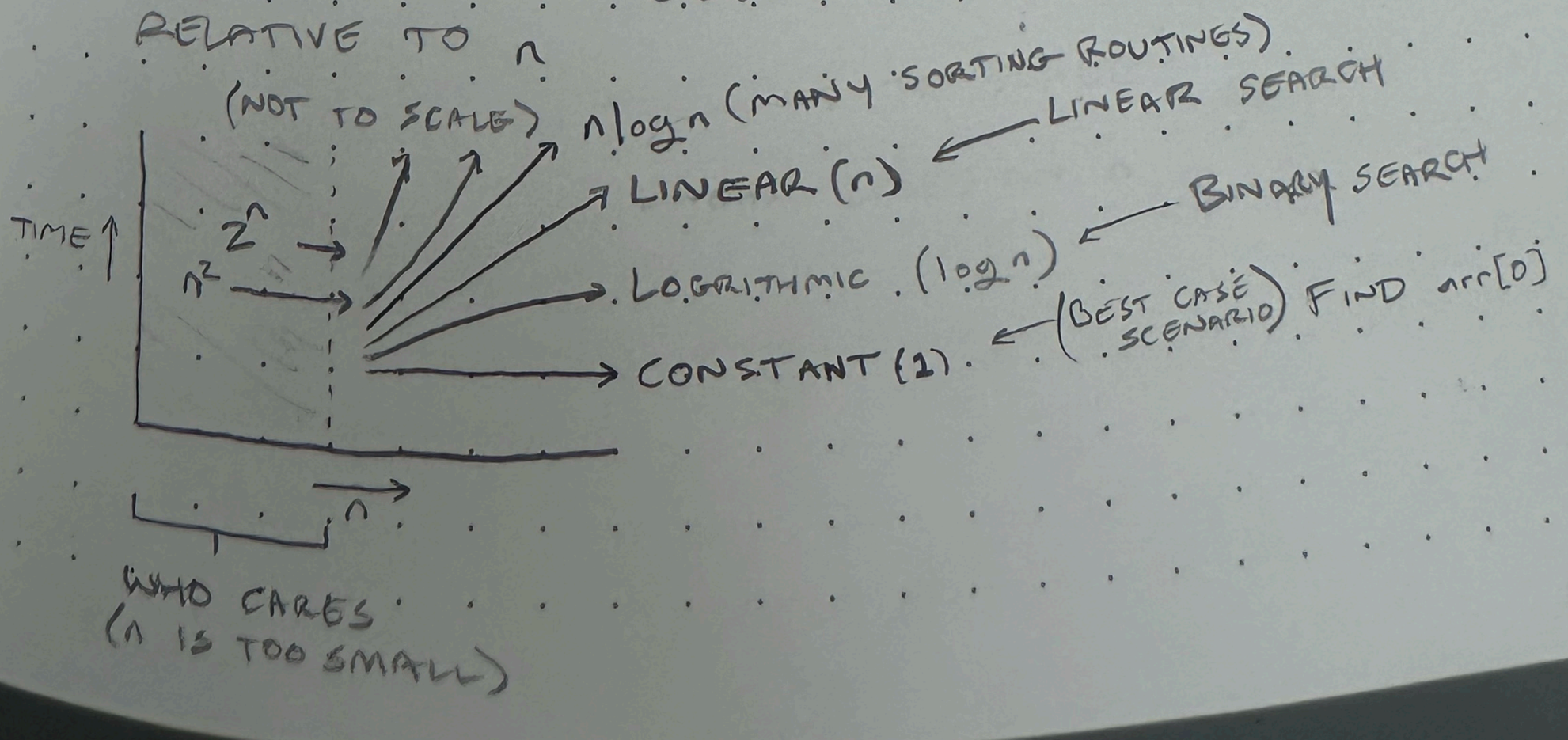
TIME COMPLEXITY: WHAT HAPPENS WHEN DATA IS LARGE?

EX ARRAY - LIKE INPUT WITH SIZE n



Q: HOW MUCH WORK DOES ALG. DO IN RELATION TO n?

A: PICK A KEY OPERATION (EX: ARRAY LOOK UP, SWAP, MULT.) "INSIDE THE LOOP". COUNT HOW MANY TIMES IT OCCURS RELATIVE TO n



TIME COMPLEXITY: CONT.

BASIC IDEA

~~$2n^2$~~
$n^2 + ~~n~~$
n^2
$\Big) \rightarrow n^2$

COEFFICIENTS AND SMALLER POWERS ARE IGNORED (TYPICALLY)

$O(\ )$
BIG-Oh

$O(1)$ - CONSTANT
$O(n)$ - LINEAR
$O(\lg)$ - LOGRITHMIC

↗ TYPICALLY ~~MEANS~~ REPRESENTS MAXIMUM TIME COMPLEXITY

* PROOFS OF O COMPLEXITY ARE IN THE BOOK

REAL WORLD SCENARIO:
CONTEST

PROGRAMMER A: 1 BILLION IPS (INSTRUCTIONS PER MINUTE); GOOD AT OPTIMIZING CODE RUNS AT 2x. SELECTS INSERTION SORT

PROGRAMMER B: 10 MILLION IPS; POOR OPTIMIZER CODE RUNS AT 50x. SELECTS MERGE SORT

GIVEN THE TASK OF SORTING 10 MILLION VALUES
PROGRAMMER B: TAKES 20 MINUTES ($O(n^2)$)
PROGRAMMER A: TAKES 2.3 DAYS ($O(n \log n)$)

O(1) INSERTION INTO A SET
O(n) LINEAR SEARCH, FIND MAX
O(log n) BINARY SEARCH
O(n log n) MERGE SORT, OTHER SORT
O(n²) SELECTION SORT, BUBBLE SORT, INSERTION SORT ← FOR SMALL DATA

HOW DO I KNOW IF MY PROGRAM RUNS AT O(x)

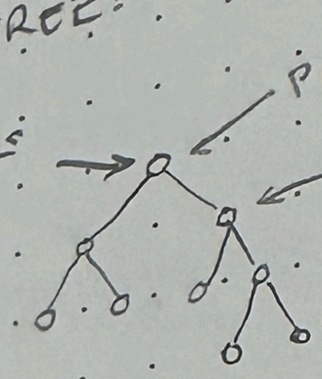
END

ING
SORT

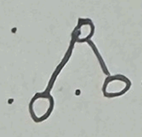
GE SORT

LVES.
))
og n))

BINARY TREE:

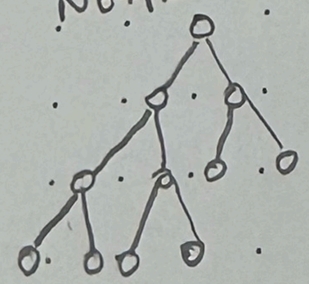


A COMPLETE BINARY TREE → EVERY LEVEL IS FULL

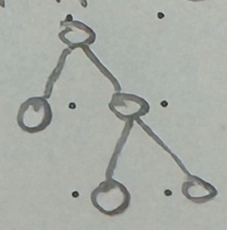


NEARLY COMPLETE IS COMPLETE IS A BINARY TREE THAT IS COMPLETE AT EVERY LEVEL EXCEPT FOR LAST.
- LAST LEVEL IS CONTIGUOUS (L TO R)

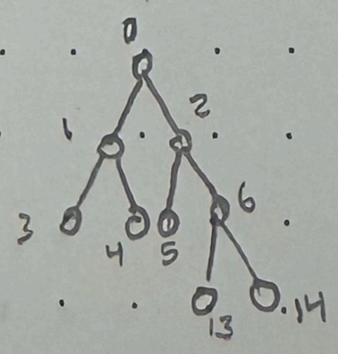
NEARLY COMPLETE

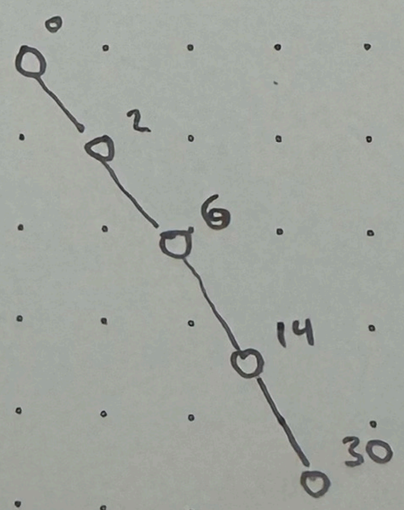


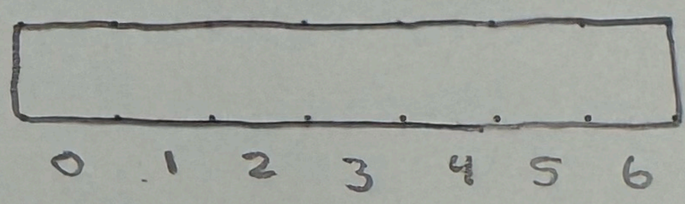
INCOMPLETE



STORING IT IN AN ARRAY







LEFT CHILD (i) = 2i + 1

RIGHT CHILD (i) = 2i + 2

PARENT (i) = (i - 1)/2

NEED AN ARRAY OF AT LEAST 30!

DSA [02/07/24]

PRIORITY QUEUE: ADDS PRIORITY TO THE QUEUE

> THE ORDERING IS NOT BASED ON TIME, RATHER BASED ON PRIORITY. AN ELEMENT CAN BE PROMOTED, BUT NOT DEMOTED

Q: HOW DO WE SUPPORT THOSE TASKS AND MAKE THEM EFFICIENT?

* A REGULAR QUEUE COULD BE IMPLEMENTED VIA LINKED LIST OR AN ARRAY.

WE WILL BUILD THE PRIORITY QUEUE ON A HEAP

HEAP → PRETTY MUCH AN ARRAY
WE MIGHT WANT TO ADD A HEAP SORT! O(lgn) (NOT SUPER USEFULL)
← "0 BASED HEAP"

HEAP - A NEARLY COMPLETE BINARY TREE THAT RESPECTS THE HEAP PROPERTY

"LAZY" DATA STRUCTURE ← GOOD

HEAP WILL BE PARTIALLY ORDERED
- REMOVE FIRST (FIGURE OUT NEW FIRST)

MAXIMUM PRIORITY QUEUE (HIGHER #, HIGHER PRIORITY)

HEAP PROPERTY: THE VALUE OF A NODE IS AT MOST THE VALUE OF ITS PARENT

$A[PARENT(i)] \geq A[i]$

IN OUR CASE, AN ARRAY IS AN EFFICIENT WAY TO STORE A HEAP.

EX HEIGHT OF NODE IN A TREE IN OUR CASE IS $O(\log n)$
WE WILL HAVE n NODES AT HEIGHT $\log n$