Which of the following is the worst algorithm?

- A. Selection Sort (with linear search)
- B. Selection Sort with binary search
- C. Selection Sort with a heap
- D. Heapsort
- E. Whatever
Removing From a PQ

• Steps
  • Store the value of root
  • Delete the right most node among the nodes with the largest depth) put its value in the root
  • while (value > value of (at least) one child )
    • Swap with a child with the smallest value
  • Return the value stored in step 1
Today’s Outline

- Binary Search Tree
  - Basics
  - Operations
  - Implementation
Searching in sorted list vs unsorted list

- Search in **unsorted** list
  - 7, 3, 12, 6, 9, 1, 15
  - Linear Search: $O(n)$
- Search in **sorted** list
  - 1, 3, 6, 7, 9, 12, 15
  - Linear Search: $O(n)$
  - Binary Search: $O(\log n)$
Data structures to store a list:

- Array/vector: $O(1)$ random access
- Linked list: $O(n)$
- Binary tree: $O(\log n)$

Data structures compatible with binary search?
Binary Trees and Orders

- Binary trees impose multiple orderings on their elements (pre-/in-/post-/level-orders)
- In particular, in-order traversal suggests a natural way to hold comparable items
  - For each node $v$ in tree
    - All values in left subtree of $v$ are $\leq v$
    - All values in right subtree of $v$ are $\geq v$
- This leads us to...
Binary Search Tree (BST)

- **Definition:**
  - A binary tree s.t. for every node \( n \) in the tree,
    \[ left < n < right \]
  where \( left = \) any node in the left subtree
  \[ right = \] any node in the right subtree
• Which of the above are BSTs?

A. 1
B. 1 and 2
C. 1 and 3
D. 1, 2, and 3
E. Whatever
Today’s Outline

• Binary Search Tree
  • Basics
  • Operations
  • Implementation
BST Operations

- BSTs will implement the OrderedStructure Interface
  - `add(E item)`
  - `contains(E item)`
  - `get(E item)`
  - `remove(E item)`
  - `iterator()` — in-order traversal

\[ \Theta(\log n) \quad \log n \leq h \leq n \]

\( O(h) \) — height of the tree
contains()

- contains(key):

  ```python
  cur = root
  if cur is Empty(): not found
  if key < cur.key: go left (cur = cur.left())
  if key > cur.key: go right
  if key == cur.key: found it.
  ```
get()

- get(key):

  cur = root
  if cur.isEmpty(): not found
  if key < cur.key: go left
  if key > cur.key: go right
  if key == cur.key: found it!
• **add(x):**

  cur = root
  if cur.isEmpty:  \[\text{add here}\]
  if key < cur.key:  go left
  if key > cur.key:  go right
  if key == cur.key:  add x at predecessor(cur)