Lecture 2!

1

Midterm exam 18-19 - MCQ 9
(1 Point)

(1 point) Consider a time-efficient algorithm to remove the middle element of a sequence S with n elements. Assume that the middle element is the element at position $n/2$, where / denotes the integer division. Example: if $S = \{1, 2, 3, 4, 5\}$, the algorithm should change S into $\{1, 2, 4, 5\}$. What is the tightest worst-case **time** complexity of removing all elements of S by calling such algorithm repeatedly until S is empty, if S is implemented using an array or a singly-linked list?

A. **array:** $\mathcal{O}(n)$ **singly-linked list:** $\mathcal{O}(n)$
B. **array:** $\mathcal{O}(n)$ **singly-linked list:** $\mathcal{O}(n^2)$
C. **array:** $\mathcal{O}(n^2)$ **singly-linked list:** $\mathcal{O}(n)$
(D) **array:** $\mathcal{O}(n^2)$ **singly-linked list:** $\mathcal{O}(n^2)$

↳ O(n) shifting ↳ O(n) traversing

2

Final exam 18-19 - MCQ6

(1 point) Consider the insertion sort algorithm. What is the state of sequence (7,1,3,6,2,5,4) after 3 complete executions of the algorithm's outer loop where the element being sorted is compared to at least one other element?

A. (1,3,7,6,2,5,4)
B. (1,2,3,6,7,5,4)
(C) (1,3,6,7,2,5,4)
D. (1,2,3,7,6,5,4)

4

How many changes to next and prev references are needed to perform an insertion into a doubly-linked list?
(1 Point)

○ 1 prev, 1 next
● 2 prev, 2 next
○ 3 prev, 3 next
○ 4 prev, 4 next

5

If a list m uses header and trailer nodes, which of the following denotes list m when it contains one element?
(1 Point)

○ m.header.next = null
○ m.header.next = trailer
● m.header.next != null and m.header.next.next != null → element, → trailer
○ m.header.next != null and m.header.next.next = null

6

Final exam 18-19 - MCQ9
(1 Point)

(1 point) Consider an algorithm to move the second and the last but one elements of a sequence S with n elements to the middle of that sequence. Example: if S has elements $(1, 2, 3, 4, 5, 6, 7, 8)$, the algorithm should change S into $(1, 3, 4, 2, 7, 5, 6, 8)$. What is the complexity of the most time-efficient algorithm for this operation when S is implemented by an array or a singly-linked list?

A. array: $\mathcal{O}(1)$ singly-linked list: $\mathcal{O}(1)$
B. array: $\mathcal{O}(1)$ singly-linked list: $\mathcal{O}(n)$
C. array: $\mathcal{O}(n)$ singly-linked list: $\mathcal{O}(1)$
(D) array: $\mathcal{O}(n)$ singly-linked list: $\mathcal{O}(n)$

7

Final exam 19-20 - MCQ 10
(1 Point)

(1 point) Consider the `clone` method below for cloning objects of class `SinglyLinkedList`.

```
1  public SinglyLinkedList<E> clone() throws CloneNotSupportedException {
2    SinglyLinkedList<E> other = (SinglyLinkedList<E>) super.clone();
3    if (size > 0) {
4      other.head = new Node<>(head.getElement(), null);
5      Node<E> walk = head.getNext();
6      Node<E> otherTail = other.head;
7      while (walk != null) {
8        Node<E> newest = new Node<>(walk.getElement(), null);
9        otherTail.setNext(newest);
10       otherTail = newest;
11       walk = walk.getNext();
12     }
13   }
14   return other;
15 }
```

Which of the following statements is **true** about method `clone`?

A. If a **node** of the clone list is replaced, the corresponding node in the original list will also change.
B. If an **element** of the clone list is replaced, the corresponding element in the original list will also change.
(C) If an **instance field** of an element in the clone list is changed, then the field of the corresponding element in the original list will also change.
D. If the **head** of the clone list is changed to point to the tail of the clone list, then the head of the original list will also point to the tail of the original list.

8

Final exam 19-20 - MCQ4

(1 point) Consider the most time-efficient algorithm that uses $\mathcal{O}(1)$ space to calculate and print the sum of every pair of consecutive elements in a sequence S with n elements. Example: if $S = [1, 2, 3, 4, 5]$, the algorithm should print the sums $3, 5, 7, 9$ corresponding to $1+2$, $2+3$, $3+4$, and $4+5$. The sequence can change in between, but its final state needs to be the same as it was before the calculations. What is the tightest worst-case **time** complexity of such algorithm when S is implemented using either a fixed-size array, or a singly-linked list without a tail reference. The list has the following ADT: `addFirst(E e)`, `addLast(E e)`, `removeFirst()`, `first()`, `size()`. Note that the ADT **does not allow** direct access to its nodes, the methods `removeFirst` and `first` return the element directly.

(A) **array:** $\mathcal{O}(n)$ **singly-linked list:** $\mathcal{O}(n)$
B. **array:** $\mathcal{O}(n)$ **singly-linked list:** $\mathcal{O}(n^2)$
C. **array:** $\mathcal{O}(n^2)$ **singly-linked list:** $\mathcal{O}(n)$
D. **array:** $\mathcal{O}(n^2)$ **singly-linked list:** $\mathcal{O}(n^2)$

9

Resit exam 17-18 - Q26 (adapted from)

What does method "operateOnMatrix" do? What is its tightest worst-case time complexity in big-Oh notation? Would the method generate the same output if it was performed only on the upper triangle of the matrix? Describe a faster solution to perform the same operation only on the upper triangle of the matrix, assuming that the elements of each row are sorted.

Method `operateOnMatrix` below performs an operation on a symmetric n by n matrix (row x column).

```
public class SymmetricMatrix {
  private int[][] matrix;
  /* ... */
  public static int[] operateOnMatrix(int x) {
    for (int i = 0; i < matrix.length; i++) {
      int j = operateOnArray(matrix[i], i, x);
      if (j > -1)
        return new int[]{i,j};
    }
    return null;
  }

  private static int operateOnArray(int[] row, int startIdx, int x) {
    for (int j = startIdx; j < row.length; j++)
      if (row[j] == x)
        return j;
    return -1;
  }
}
```

Entries in a symmetric matrix are symmetric with respect to the main diagonal. Example: $M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 6 \end{pmatrix}$

$O(n^2)$ time complexity
$O(n \log n)$ -> binary search on each row

Lecture 2.2

1

Midterm exam 19-20 - MCQ 12
(1 Point)

(1 point) Consider that we insert the elements of an array of elements $[1, 2, 3, 4, 5, 6, 7, 8]$ into a stack and a queue as follows. Elements at even indices are inserted into the stack using `push`, while elements at odd indices are inserted into the queue using `enqueue`. The array is processed from left to right (indices 0 to 7), and both stack and queue are initially empty. The elements are then retrieved as follows: first all elements from the stack using a series of `pop` operations, then all elements from the queue are retrieved using a series of `dequeue` operations. In what order are the elements of the original array returned?

A. $1, 3, 5, 7, 2, 4, 6, 8$
B. $1, 3, 5, 7, 8, 6, 4, 2$
C. $7, 5, 3, 1, 8, 6, 4, 2$
(D) $7, 5, 3, 1, 2, 4, 6, 8$

2

Midterm exam 19-20 - MCQ 13
(1 Point)

(1 point) When implementing a stack using an array or a singly-linked list, where should the top of the stack be located such that the `push` (insertion) and `pop` (removal) operations are as efficient as possible? Note that the array is filled in the usual way, with any remaining empty space at the end.

A. **array:** beginning **singly-linked list:** head
B. **array:** beginning **singly-linked list:** tail
(C) **array:** end **singly-linked list:** head
D. **array:** end **singly-linked list:** tail

3

Final exam 18-19 - MCQ 3
(1 Point)

(1 point) Consider a deque `d` containing elements $(1, 2, 3, 4, 5, 6)$, in this order, and an empty stack `s`. We first execute the following block of instructions **three times**:

```
1 s.push(d.last());
2 s.push(d.removeLast());
3 d.removeLast();
```

Then, we remove all elements from `s` using a series of `s.pop()` operations. What elements are returned and in what order?

A. (1,2,3,4,5,6)
B. (4,4,5,5,6,6)
(C) (2,2,4,4,6,6)
D. (6,6,4,4,2,2)

5

Midterm exam 19-20 - MCQ 6
(1 Point)

(1 point) Consider that an operation without array resizing in a given data structure takes $O(\log_2 n)$ time, where n is the number of elements. The data structure is implemented using a dynamic array that grows when needed by adding $n/2$ extra positions. What is the tightest worst-case **time** complexity of performing the mentioned operation n times?

A. $\mathcal{O}(\log_2 n)$
B. $\mathcal{O}(n)$
(C) $\mathcal{O}(n \log_2 n)$
D. $\mathcal{O}(n^2)$

resizing = $O(n)$
$n\, O(\log_2 n) = O(n \log_2 n)$

6

Final exam 17-18 - MCQ 3
(1 Point)

(1 point) Consider a dynamic array, whose capacity C is incremented by inc whenever the array is full (new capacity is $C + inc$). Which of the following values of inc **does not** lead to amortized time complexity $\mathcal{O}(n)$ for n push operations (insertions at the end)? Consider that the initial capacity is given by C_0, where $1 \leq C_0 \ll n$ (C_0 is a positive integer, much smaller than n).

A. $inc = C$
B. $inc = \lceil C/2 \rceil$
C. $inc = C + 1$
(D) $inc = C_0$

Lecture 3.1

4

Midterm exam 19-20 - MCQ 19
(1 Point)

(1 point) Which of the following statements about iterators is **true**?

A. The construction of a lazy iterator takes $\mathcal{O}(n)$ time.
B. The construction of a snapshot iterator uses $\mathcal{O}(1)$ space.
C. The behavior of a lazy iterator is not affected by changes to the primary collection on which the iterator operates.
(D) The Java expression `for(Integer i : collection)`, where the class of `collection` implements the `Iterable` interface, creates an iterator for the elements in list `collection`.

1

Midterm exam 18-19 - MCQ 17
(1 Point)

17. (1 point) Which of the following statements is **true** about positional lists?

A. A Position `p` of a linked positional list enables direct access to the previous and next positions using `p.prev` and `p.next`.
B. The methods that add nodes to a linked positional list return nodes of the list.
(C) A `Position` is an abstraction used to encapsulate the actual nodes and prevent unauthorized modifications of the list.
D. Array-based positional lists are more space-efficient than standard array-based lists.

6

Final exam 17-18 - MCQ 7
(1 Point)

(1 point) Consider the same class `MyNestedClass` from question 6. Which property and underlying data structure are correct for class `MyNestedClass`?

(A) Lazy, on an array-based list.
B. Snapshot, on an array-based list.
C. Lazy, on a linked list.
D. Snapshot, on a linked list.

2

Midterm exam 18-19 - MCQ 19
(1 Point)

(1 point) Consider the implementation of methods `printElements1` and `printElements2` below and assume that the list `elements` given as input contains the following elements: $\{10, 50, 100, 500, 1000\}$.

```
1  public static void printElements1(List<Integer> elements) {
2    int i = 0;
3    while(elements.iterator().hasNext() && i < 3) {
4      System.out.println(elements.iterator().next());
5      i++;
6    }
7  }
8
9  public static void printElements2(List<Integer> elements) {
10   int i = 0;
11   Iterator<Integer> iterator = elements.iterator();
12   while(iterator.hasNext() && i < 3) {
13     System.out.println(iterator.next());
14     i++;
15   }
16 }
```

Which elements are printed by calling method `printElements1` above, and in what order?

A. $\{10, 10, 10\}$ ← method1
B. $\{50, 50, 50\}$
C. $\{10, 50, 100\}$ ← method2
D. $\{10, 50, 100, 500, 1000\}$

Lecture 32.

5

Final exam 17-18 - MCQ6
(1 Point)

(1 point) Consider the implementation of class `MyNestedClass` below.

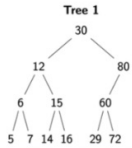
```
public class MyClass<E> {
  private int size;
  private E[] list;
  /* ... */

  private class MyNestedClass implements MyInterface<E> {
    private int j = 0;
    private boolean removable = false;

    public boolean hasNext() {
      return j < size;
    }

    public E next() throws NoSuchElementException {
      if (j == size) throw new NoSuchElementException("No next element");
      removable = true;
      return list[j++];
    }

    public void remove() throws IllegalStateException {
      if (!removable) throw new IllegalStateException("Nothing to remove");
      this.remove(j-1);
      j--;
      removable = false;
    }
  }
}
```

Which type of data structure does class `MyNestedClass` above implement?

- (A) Iterator.
- B. Positional list.
- C. Array list.
- D. Singly-linked list.

Consider the following trees **Tree 1** and **Tree 2**.

Tree 1



8

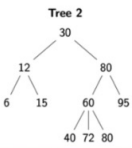
Midterm exam 17-18 - MCQ 20 (answer based on trees from MCQ19 above)
(1 Point)

(1 point) Consider the result $\{5, 7, 6, 14, 16, 15, 12, 29, 72, 60, 80, 30\}$ of performing a traversal through **Tree 1**. Which kind of traversal was performed?

- A. In-order traversal
- B. Breadth-first traversal
- C. Pre-order traversal
- (D) Post-order traversal

Consider the following trees **Tree 1** and **Tree 2**.

Tree 2



1

Midterm exam 17-18 - MCQ 13
(1 Point)

(1 point) Consider an implementation of a priority queue with a sorted list. Which operation establishes the ordering of the keys?

- (A) `insert`
- B. `removeMin`
- C. `min`
- D. A priority queue needs to be explicitly sorted using the `sort` method.

9

Midterm exam 17-18 - MCQ 21 (answer based on trees from MCQ19 above)
(1 Point)

(1 point) Consider the result $\{30, 12, 80, 6, 15, 60, 95, 40, 72, 80\}$ of performing a traversal through **Tree 2**. Which kind of traversal was performed?

- A. In-order traversal
- (B) Breadth-first traversal
- C. Pre-order traversal
- D. Post-order traversal

2

Final exam 17-18 - MCQ 8
(1 Point)

(1 point) Consider three implementations of a priority queue ADT respectively using: an unsorted list, a sorted list, and a heap. For each implementation, which operation(s) consider(s) the values of the keys?

- A. **unsorted list:** insert **sorted list:** removeMin **heap:** both
- (B) **unsorted list:** removeMin **sorted list:** insert **heap:** both
- C. **unsorted list:** removeMin **sorted list:** insert **heap:** removeMin
- D. **unsorted list:** insert **sorted list:** removeMin **heap:** insert

4

Midterm 17-18 - MCQ 15
(1 Point)

(1 point) Consider the bottom-up construction of an array-based heap. What happens after the first phase in which all entries are added to the array?

- A. Down-heap bubbling from the root node.
- B. Down-heap bubbling from all nodes except the ones in the last level, starting from the root node and ending at the last node in the last but one level.
- (C) Down-heap bubbling from all nodes except the ones in the last level, starting from the last node in the last but one level and ending up at the root node.
- D. Up-heap bubbling from the root node.

5

Midterm exam 19-20 - MCQ 9
(1 Point)

(1 point) Consider the insertion of a new entry into a heap. What happens when the heap order property is not satisfied after the new node is inserted into the heap?

- A. Nothing.
- B. Heapify.
- C. Down-heap bubbling.
- (D) Up-heap bubbling.

8

Final exam 18-19 - MCQ 2
(1 Point)

(1 point) Consider the time complexity of `insert` in a min-heap with n items, implemented using a dynamic array that grows when full from capacity C to capacity $C + \frac{C}{3}$. Which statement is **correct**?

- A. The amortized time complexity of one `insert` operation is $\mathcal{O}(1)$.
- B. The time complexity of one `insert` operation is always $\mathcal{O}(\log n)$.
- C. Performing n `insert` operations takes $\mathcal{O}(n)$ time.
- (D) Performing n `insert` operations takes $\mathcal{O}(n \log n)$ time.

6

Midterm exam 19-20 - MCQ 10
(1 Point)

(1 point) Consider the array of elements $[12, 10, 9, 5, 6, 1]$ in the given order. What is the content of the array after applying the heapify algorithm to build a min-heap?

- A. $[1, 5, 6, 9, 10, 12]$
- B. $[1, 5, 12, 10, 6, 9]$
- (C) $[1, 5, 9, 10, 6, 12]$
- D. $[1, 5, 6, 10, 9, 12]$

Midterm exam 19-20 - MCQ 11
(1 Point)

(1 point) What is the time complexity of the heapify algorithm?

- A. $\mathcal{O}(\log_2 n)$
- (B) $\mathcal{O}(n)$
- C. $\mathcal{O}(n \log_2 n)$
- D. $\mathcal{O}(n^2)$

Lecture 4.1.

1

Midterm exam 17-18 - MCQ 23
(1 Point)

What is the tightest big-Oh time complexity of this method?

23. Consider the following implementation of `methodX`. Let the time complexities of methods `insert` and `removeMin` be $\mathcal{O}(1)$ and $\mathcal{O}(k)$, respectively, where k denotes the number of entries in priority queue `P`. `ArrayList` methods used in this code have the following time complexities: `size` and `add` are $\mathcal{O}(1)$, `remove` as used in the code is $\mathcal{O}(1)$ since it always removes at the end. Assume that the priority queue `P` is empty when `methodX` is called.

```
public static <E> void methodX(ArrayList<E> S, PQ<E, Object> P) {
  int n = S.size();
  for (int j = n-1; j >= 0; j--) {
    E element = S.remove(j);
    P.insert(element, null);
  }
  for (int j = 0; j < n; j++) {
    E element = P.removeMin().getKey();
    S.add(element);
  }
}
```

Selection sort

↳ O(n)

total: O(n²)

3

Which of these sorting algorithms are asymptotically faster than **methodX** of question 1?
(1 Point)

- [ ] Insertion sort
- [ ] Heap sort
- [x] Merge sort O(nlogn)
- [ ] Sort using a sorted list priority queue

5

Final 18-19 - MCQ 13
(1 Point)

(1 point) Which of these statements is **true** about the insertion sort algorithm?

A. When processing element x at index i in the outer loop, the subarray between indexes 0 and $i-1$ is unsorted.
B. The outer loop scans right to left, one element x at a time. The inner loop finds the final position for x among all elements and inserts it by shifting other elements as needed.
C. The outer loop scans left to right, one element x at a time. The inner loop finds the final position for x among all elements and inserts it by shifting other elements as needed.
D. The outer loop scans left to right, one element x at index i a time. The inner loop compares all elements in the subarray between indexes 0 and $i-1$ to x and shifts them to the right if not in order. Element x is inserted in its position within indexes 0 and i.

4

Final 18-19 - MCQ 6
(1 Point)

(1 point) Consider the insertion sort algorithm. What is the state of sequence (7,1,3,6,2,5,4) after 3 complete executions of the algorithm's outer loop where the element being sorted is compared to at least one other element?

A. (1,3,7,6,2,5,4)
B. (1,2,3,6,7,5,4)
C. (1,3,6,7,2,5,4)
D. (1,2,3,7,6,5,4)

7

Midterm 19-20 - MCQ 17
(1 Point)

(1 point) Consider the implementation of method `sort` below.

```
1  public void sort(int a[]) {
2    for (int i = 0; i < a.length-1; i++) {
3      int idx = i;
4      for (int j = i+1; j < a.length; j++)
5        if (a[j] < a[idx])
6          idx = j;
7      swap(a, idx, i);
8    }
9  }
```

Which specific sorting algorithm does method `sort` above implement?

A. Selection sort.
B. Insertion sort.
C. Heap sort.
D. Merge sort.

6

Resit 18-19 - MCQ 11
(1 Point)

(1 point) Consider that the selection sort algorithm is applied to sort the sequence (1,7,6,2,8,4,5,3) in increasing order. What is the state of the sequence after a number of iterations resulting in 3 swaps?

A. (1,2,3,7,8,4,5,6)
B. (1,2,6,7,8,4,5,3)
C. (1,2,3,4,8,7,5,6)
D. (1,2,3,4,5,7,8,6)

8

Final 19-20 - MCQ 13
(1 Point)

(1 point) Consider the sorting of sequence $[4, 7, 12, 9, 1, 3]$ in increasing order with the in-place heap sort algorithm using an array-based heap. What is the content of the array after the first two removals from the heap (including any necessary bubbling operations)?

A. $[1, 7, 4, 3, 9, 12]$
B. $[7, 1, 4, 3, 9, 12]$
C. $[7, 3, 4, 1, 9, 12]$
D. $[1, 3, 4, 7, 12, 9]$

Lecture 4.2.

9

Midterm 19-20 - MCQ 16
(1 Point)

(1 point) Consider the application of the merge sort algorithm to sort the sequence of elements $[12, 10, 9, 5, 6, 1]$ in increasing order. What are the sequences found at level 2 of the merge sort recursion tree after merging the sequences returned by level 3? Remember that the root is at level 0 of this recursion tree.

A. [12, 10, 9], [5, 6, 1]
B. [9, 10, 12], [1, 5, 6]
C. [12], [9, 10], [5], [1, 6]
D. [12], [10, 9], [5], [6, 1]

10

Final 20-21 - MCQ 10
(1 Point)

Consider the main method of the bottom up merge sort algorithm below:

```
1  public static void mergeSortBottomUp(int[ ] orig) {
2    int n = orig.length;
3    int[ ] src = orig;
4    int[ ] dest = (int[ ]) new int[n];
5    int[ ] temp;
6    for (int i = 1; i < n; i *= 2) {
7      for(int j = 0; j < n; j += 2*i) {
8        merge(src, dest, j, i);
9      }
10     temp = src; src = dest; dest = temp;
11   }
12   if (orig != src)
13     System.arraycopy(src, 0, orig, 0, n);
14 }
```

How many calls to the auxiliary method `merge` are performed when `mergeSortBottomUp` is applied to the array `[1, 2, 4, 5, 6, 6, 9, 9, 9]`?

Select one answer

- [ ] 8 calls
- [ ] 9 calls
- [ ] 10 calls
- [x] 11 calls

1

Resit 18-19 - MCQ 10
(1 Point)

(1 point) What is the recurrence equation with $n > 1$ for the **worst-case** of the quicksort algorithm, where n denotes the input size and c_1 and c_2 are positive integer constants?

A. $T(n) = T(n-2) + c_1 n + c_2$
B. $T(n) = T(n-1) + c_1 n + c_2$
C. $T(n) = T(n/2) + c_1 n + c_2$
D. $T(n) = 2T(n/2) + c_1 n + c_2$

2

Resit 20-21 - MCQ 17
(1 Point)

Consider that you are sorting an array of integers using the in-place quick sort algorithm. After you perform the first partitioning, the array is in the following state: (3,6,8,2,7,9,12,11). Which statement is correct?

- The pivot could have been either the 7 or the 9.
- The pivot could have been the 7, but not the 9.
- **The pivot could have been the 9, but not the 7.**
- Neither the 7 nor the 9 could have been the pivot.

4

Which algorithm has the lowest **worst-case** big-Oh time complexity to find the median element in an unordered array with n elements? Note: the median is the (n/2)-th element where / denotes integer division. Select one answer.
(1 Point)

smallest/largest

- Binary search
- Quick sort followed by retrieval of the element at index n/2-1.
- Building of an array-based heap followed by retrieval of the element at index n/2-1.
- **Building of a heap followed by removal of the first (n/2)-th elements.**

5

Which algorithms have the lowest **expected** time complexity to find the median element in an unordered array with n elements? Note: the median is the (n/2)-th element where / denotes integer division. Select all answers that apply.
(1 Point)

smallest/largest

- Binary search.
- **Quick sort followed by retrieval of the element at index n/2-1.**
- Building of an array-based heap followed by retrieval of the element at index n/2-1.
- **Building of a heap followed by removal of the first (n/2)-th elements.**

7

What is the time complexity of the partition step in the in-place quick sort algorithm when applied to a subarray of size k?
(1 Point)

- $O(1)$
- $O(\log k)$
- **$O(k)$**
- $O(k \log k)$

8

What is the tightest worst-case **space** complexity of the in-place quick sort algorithm when applied to an array with n elements?
(1 Point)

- $O(\log n)$
- **$O(n)$**
- $O(n \log n)$
- $O(n^2)$

3

Resit 19-20 - MCQ 12
(1 Point)

Consider that we apply the **in-place quicksort algorithm** to sort the array $(8, 9, 5, 1, 6, 2, 3, 4, 7)$ in increasing order.

The pivot is chosen as the middle element as follows: when partitioning the portion of the array between indices `a` and `b`, pick the element at index `(a+b)/2` where `/` denotes integer division, swap it with the last element at index `b`, and then perform the partition algorithm as usual using the last element as the pivot. Example: to partition $(8, 9, 5, 1, 6, 2, 3, 4, 7)$ with indices `a = 0` and `b = 8`, first swap the pivot element `6` (at index `(0+8)/2 = 4`) with the last element `7` resulting in sequence $(8, 9, 5, 1, 7, 2, 3, 4, 6)$. After this, the partition algorithm is executed based on the pivot element `6` (now at index `8`).

What is the content of the array after the **first partitioning** and before the recursive calls at **level 2** of the quicksort recursion tree? Remember that the root is at level 0 of this recursion tree. Also note that the first partitioning at level 2 is performed on the portion of the array corresponding to the left partition obtained at level 1. Here we refer to the in-place partitioning algorithm seen in the lectures/book.

- (4,3,2,1,5,6,9,8,7)
- **(2,1,3,4,5,6,9,8,7)**
- (1,2,3,4,5,6,9,8,7)
- (1,2,3,4,5,6,7,8,9)

6

Which of the following statements about quick sort are true? Select all that apply.
(1 Point)

- Choosing the second element as the pivot guarantees that we avoid the worst-case.
- **Quick sort does the work of sorting in the partition step.**
- Quick sort makes three recursive calls, respectively for the elements less than, equal to, and larger than the chosen pivot.
- **In the partition step of the in-place quick sort algorithm, elements that are equal to the pivot can end up on either side of the pivot.**

9

Final 17-18 - MCQ 9
(1 Point)

(1 point) Which of the following statements about comparison-based sorting algorithms is **true**?

A. Selection sort is faster than insertion sort for all inputs.
B. Quick sort is always the fastest amongst comparison-based sorting algorithms.
C. Heap sort cannot be implemented in-place.
D. Quick sort and merge sort follow the divide-and-conquer paradigm.

10

Final 17-18 - MCQ 10
(1 Point)

(1 point) What would be the fastest algorithm to sort a sequence S of 50 nearly sorted integers, with only 2 elements out of place?

A. Selection sort
B. Insertion sort
C. Merge sort
D. Quick sort

Sorting a sequence of n elements where only 2 are out ot place:

A. Selection sort would still take $O(n^2)$, which could be faster than an $O(n \log n)$ algorithm for a small sequence but not faster than insertion sort.
B. Insertion sort takes $O(n)$ time in this case.
C. Merge sort can be optimized for sorted sequences, but it always has quite the overhead due to recursion and not being in-place, which makes it slower for smaller sequences.
D. Quick sort will take expected $O(n \log n)$ time.

Lecture 6.1

1

Final exam 19-20 - MCQ 14
(1 Point)

(1 point) Consider the following method csort, which implements a sorting algorithm. The input array is guaranteed to contain integer values between 0 and k. O(n+k)

```
1  public static void csort(int[] array, int k) {
2    int temp[] = new int[k + 1];
3
4    for (int e : array)
5      temp[e]++;
6
7    int ndx = 0;
8    for (int i = 0; i < temp.length; i++)
9      while (temp[i] > 0) {
10       array[ndx] = i;
11       ndx++;
12       temp[i]--;
13     }
14 }
```

csort is O(n) n=10^6 k=256 k<<n

Which of the following statements about method csort and the sorting algorithm it implements is **true**?

A. It is a comparison-based sorting algorithm.
B. It is an in-place sorting algorithm with $\mathcal{O}(1)$ space complexity.
C. It runs in $\mathcal{O}(n)$ time, where n is the length of the input array.
D. At each index, the array temp accumulates the number of elements from the original input array that are identical to such index.

2

Final exam 19-20 - MCQ 15 (answer based on the code from the previous question).
(1 Point)

(1 point) Consider again the method given in question 14. Between method csort and the algorithm merge sort, which one has the fastest running time in each of the following two situations? First, sorting 1 million bytes. Second, sorting 100 integers in the range between 0 and 1 million.

A. first: merge sort; second: merge sort
B. first: merge sort; second: csort
C. first: csort; second: csort
D. first: csort; second: merge sort

csort is O(n^3) n=10^6 k=10^6*n^3 k>>n

3

(1 Point)

What is the content of the following sequence of elements [329, 457, 657, 839, 436, 720, 355] after two iterations of the LSD radix sort algorithm using digits as individual keys?

- [720, 329, 436, 839, 355, 457, 657]
- [720, 355, 436, 457, 657, 329, 839]
- [329, 355, 457, 436, 657, 720, 839]
- [329, 355, 436, 457, 657, 720, 839]

4

(1 Point)

How many non-empty buckets are formed during the first iteration of the MSD radix sort algorithm when applied to the sequence [329, 457, 657, 839, 436, 720, 355] using decimal digits as individual keys?

- 4
- 5
- 6
- 7

5

Final exam 19-20 - MCQ 4
(1 Point)

(1 point) Which of the following statements on sorting algorithms is **false**?

A. Insertion sort can run in linear time if the input sequence is nearly sorted.
B. Radix sort can be slower than $\mathcal{O}(n \log n)$ time sorting algorithms if the keys to be sorted are large (e.g. when the number d of elementary keys of each composite key is similar to n).
C. Merge sort can only be applied to sequences that fit into the main memory.
D. In the MSD radix sort variant that sorts from most to least significant keys, bucket sort needs to be applied recursively within each bucket defined in the previous iteration.

6

Final exam 19-20 - MCQ 7
(1 Point)

Which algorithm is implemented by method expertK above?

7. (1 point) Consider the implementation of method expertK below.

```
1  public static int expertK(int[] array, int k) {
2    return expertK(array, 0, array.length-1, k);
3  }
4
5  private static int expertK(int[] array, int a, int b, int k) {
6    if (a == b) return array[a];
7    int left = a, right = b-1, choice = array[b];
8
9    while (left <= right) {
10     while (left <= right && array[left] < choice) left++;
11     while (left <= right && array[right] >= choice) right--;
12     if (left <= right) {
13       swap(array, left, right);
14       left++;
15       right--;
16     }
17   }
18   swap(array, left, b);
19
20   if (k <= left-a) return expertK(array, a, left-1, k);
21   else if (k <= left-a+1) return array[left];
22   else return expertK(array, left+1, b, k-left+a-1);
23 }
```

- Selection sort.
- Quick sort.
- Median find.
- Quick select.

7

Final exam 19-20 - MCQ 16
(1 Point)

(1 point) Which of the following statements about sorting and selection algorithms is **true**?

A. Heap sort, merge sort, and bucket sort are naturally stable sorting algorithms.
B. The time complexity of insertion sort is $\mathcal{O}(n+m)$, where n is the size of the input sequence and m is the number of inversions. In the worst-case, this can be $\mathcal{O}(n^2)$.
C. LSD radix sort may not need to process all the individual keys of a composite key in order to determine the final ordering.
D. The quick-select, merge sort, and quicksort algorithms follow the divide-and-conquer paradigm.

9

Resit exam 19-20 - MCQ 2
(1 Point)

Consider that we would like to find and print duplicate elements in an **unsorted** sequence of integers. For example, given the sequence (3, 2, 4, 3), we would like to print the number 3.

Which of the following data structures or algorithms would lead to the most time-efficient solution for this task, considering tightest **average or expected time** complexity in big-Oh notation?

- Quick sort.
- Quick select.
- Nested loops to make comparisons between all pairs of elements.
- List-based priority queue.

10

Resit exam 17-18 - MCQ 11
(1 Point)

11. (1 point) Which algorithm has the best **expected** time complexity to find the median element of an unordered sequence S with n elements, assuming that n is odd?

A. Binary search.
B. Quick select.
C. Quick sort followed by retrieval of the $\lceil n/2 \rceil^{th}$ element.
D. Build a min-heap and extract $\lceil n/2 \rceil$ elements. The median is the $\lceil n/2 \rceil^{th}$ element.

Lecture 6.2

Map$\langle K, V \rangle$	Unsorted arraylist	Sorted arralyist	Arrays based using hashing
get(K key)	$\mathcal{O}(n)$	$\mathcal{O}(log(n))$	$\mathcal{O}(1)$ (expected)
put(K key, V value)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$ (expected)
remove(K key)	$\mathcal{O}(n)$	$\mathcal{O}(n)$	$\mathcal{O}(1)$ (expected)

Would it be possible to implement hashmaps without a key? And without a value?

(a) Yes, we can implement them without a key or value (but we still need at least one of them)

(b) We can implement them without a key, but not without a value

(c) We cannot implement them without a key, but we could implement them without a value

(d) No, we need both the key and the value to implement them

If we have the worst hash code possible, can we fix this with a good compression function?

(a) Yes, we can still have a good hash function

(b) Partially, we can have a decent hash function, but it will not be perfect

(c) No, but we can make it a little better

(d) No, not at all

What is true about hash codes?

(a) If x.equals(y), then x.hashCode() == y.hashCode()

(b) If x.hashCode() == y.hashCode(), then x.equals(y)

(c) Both A & B are true

(d) Neither A nor B is true problem

If we use the same letter to integer conversion as in the video, to what would the name Sara hash if we use a polynomial hash code with $a = 5$?

(a) 175

(b) 693

(c) 2335

(d) 2533

S=18
a=0
r=17
$0.5^0 + 17.5^1 + 0.5^2 + 18.5^3$

Which core function(s) of a map do(es) not work anymore if we do not replace a deleted entry with a "defunct" object?

(a) get
(b) put
(c) remove
(d) get and put
(e) get and remove
(f) put and remove
(g) get, put and remove

Given an array-based map using hashing, with Separate Chaining as collision-handling scheme, how large should the array be to ensure a time complexity of $\mathcal{O}(1)$ for the core operations (choose the last option from the correct answers)?

(a) $N > n$

(b) $N \geq n$

(c) N is $\Omega(n)$

(d) N is prime

(e) N is $\Omega(1)$

Say that for some reason, we want to implement a hashmap where we rehash once our load factor λ reaches 1.1. Which collision-handling scheme should we choose?

(a) Separate Chaining

(b) Linear probing

(c) It doesn't matter

Say we have an empty hashmap with capacity N. Now, we add n (where $n << N$) elements to the hashmap, which happen to be in an order with decreasing keys, and each key appears twice. For example, we first add an element with key 8, then an element with key 8, then an element with key 7, then an element with key 7, then an element with key 6, etc. What is the time complexity of adding those n elements, using separate chaining and using open addressing?

(a) Separate Chaining: $\mathcal{O}(n)$, Open addressing: $\mathcal{O}(n)$

(b) Separate Chaining: $\mathcal{O}(n)$, Open addressing: $\mathcal{O}(n^2)$

(c) Separate Chaining: $\mathcal{O}(n^2)$, Open addressing: $\mathcal{O}(n)$

(d) Separate Chaining: $\mathcal{O}(n^2)$, Open addressing: $\mathcal{O}(n^2)$

Get

- If the given key is null, return null
- Calculate the hash of the given key
- Find the list belonging to that hash
- For each entry in that list:
  - If the entry has the given key, return the value of that entry
- If it isn't found, return null

- If the given key is null, return null
- Calculate the hash of the given key
- For that place in the list, and then for the places following it:
  - If the entry is null, the key is not there so return null
  - If the entry has the given key, return the value of that entry
  - If the entry is a different key or defunct, continue searching
- If it isn't found, return null

Remove

- If the given key is null, return false
- Calculate the hash of the given key
- Find the list belonging to that hash
- For each entry in that list:
  - If the entry has the given key, remove that entry and return true
- If it isn't found, return false

- If the given key is null, return false
- Calculate the hash of the given key
- For that place in the list, and then for the places following it:
  - If the entry is null, the key is not there so return false
  - If the entry has the given key, set it DEFUNCT and return true
  - If the entry is a different key or defunct, continue searching
- If it isn't found, return false

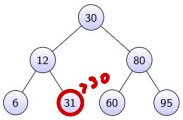
Put

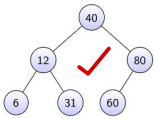
- If the given key is null, return false
- Calculate the hash of the given key
- Find the list belonging to that hash
- For each entry in that list:
  - If the entry has the given key, replace its value with the given value and return true
- If it isn't found, add an entry to the list with given key and given value and return true

- If the given key is null, return false
- Calculate the hash of the given key
- For that place in the list, and then for the places following it:
  - If the entry is null, put the given key and given value in that entry, and return true
  - If the entry has the given key, put the given key and given value in that entry, and return true
  - If the entry is DEFUNCT, ?
  - If the entry is a different key, continue searching
- We cannot place the entry, so return false

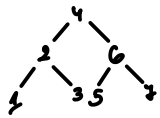
Lecture 7.1.

Are the following trees binary search trees?



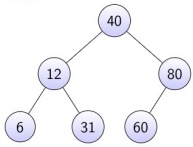


In what order do we need to put the elements in the tree to create the most balanced tree?

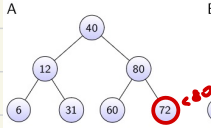


(a) 1,2,3,4,5,6,7
(b) 2,4,6,1,3,5,7
(c) 2,6,4,1,3,5,7
(d) 2,5,3,6,1,7,4
(e) 4,2,1,3,5,6,7
(f) 4,6,1,2,3,5,7
(g) 4,6,7,2,3,5,1
(h) 7,6,5,4,3,2,1

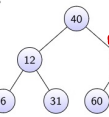
If we add 72 to the tree below, what tree do we get?



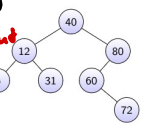
A



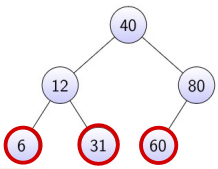
B



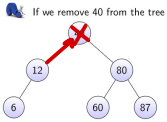
C



The last operation I executed on the tree below was a put operation. Which of the elements of the tree could that have been?

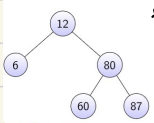


If we remove 40 from the tree below, what tree do we get?



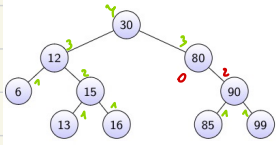
A

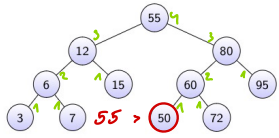
The last operation I executed on the tree below was a remove operation. What possible keys could the removed element have had?



$\forall x \in \mathbb{N},\ x \neq 6, 12, 60, 80, 87$

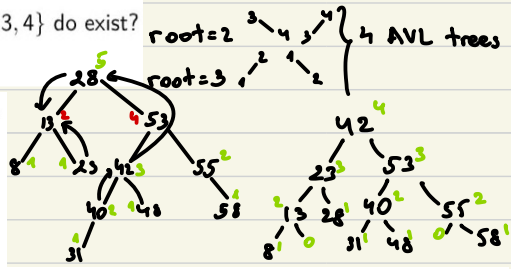
Are the following trees AVL trees?



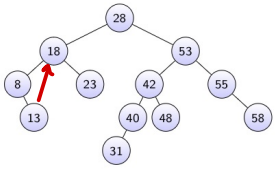


How many different AVL trees with keys $\{1, 2, 3, 4\}$ do exist?

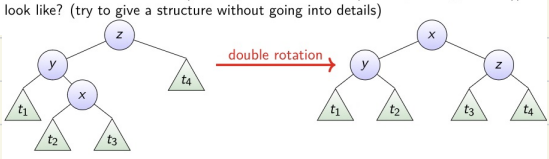
(a) 1 (b) 2 (c) 4 (d) 6 (e) 8 (f) 14



When deleting 18 from the following AVL tree, how many tri-node restructurings are needed?
(a) none (b) one (c) two (d) three



If you were asked to implement the trinode restructure below of a binary search tree, how would the method (probably called restructure(Node x, Node y, Node z)) look like? (try to give a structure without going into details)

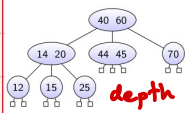


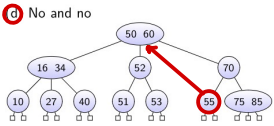
Answer: 2 rotations. First: rotate(x,y), then: rotate(x,z). A separate rotation method could just update the childs of the nodes involved in the rotation.

Lecture 7.2.

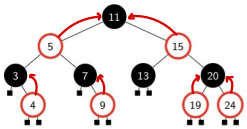
Are the following trees (2,4) trees?

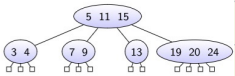
(a) Yes and yes (b) Yes and no (c) No and yes (d) No and no





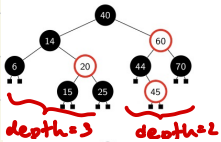
Draw the corresponding (2,4) tree

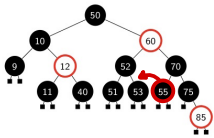




Are the following trees red-black trees?

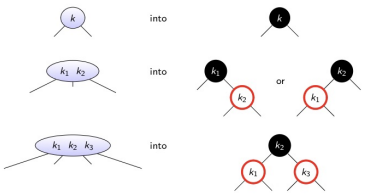
(a) Yes and yes (b) Yes and no (c) No and yes (d) No and no



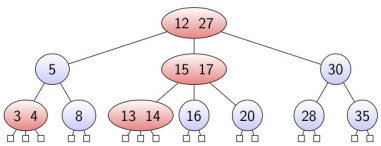


How to transform a (2,4) tree into a corresponding red-black tree in general?

By performing the following transformations:



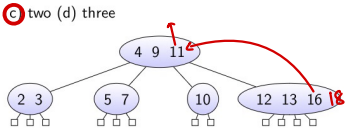
How many red-black trees correspond to the following (2,4) tree?



For each 3-node we have two choices. The above tree contains four 3-nodes, so $2^4 = 16$ red-black trees correspond to this (2,4) tree

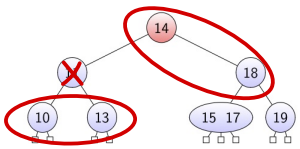
When inserting 18 into the following tree, how many overflows are caused?

(a) none (b) one (c) two (d) three



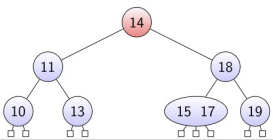
Which rebalancing operations need to performed when removing 11?

(a) none (b) one transfer (c) one fusion (d) two fusions (e) one fusion and one transfer



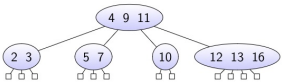
Which rebalancing operations need to performed when removing 11 from the Red-Black Tree corresponding to this (2,4) Tree?

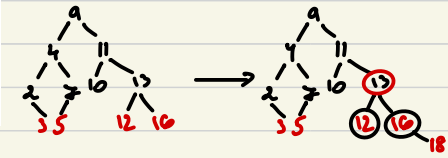
(a) Nothing, in the Red-Black tree, removing 11 wouldn't be a problem
(b) A single tri-node restructuring
(c) A single recoloring
(d) 2 tri-node restructurings
(e) A tri-node restructuring and a recoloring
(f) 2 recolorings

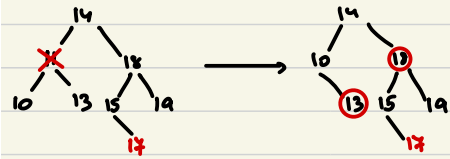


When inserting 18 into the Red-Black tree that corresponds to this (2,4) Tree, what happens?

(a) Nothing, in the Red-Black tree, inserting 18 wouldn't be a problem
(b) A single tri-node restructuring
(c) A single recoloring
(d) 2 tri-node restructurings
(e) A tri-node restructuring and a recoloring
(f) 2 recolorings







Lecture 8.1

$0 \le m \le \frac{n(n-1)}{2}$

$\sum_{v \in V} deg(v) = 2m$

$\sum_{v \in V} indeg(v) = \sum_{v \in V} outdeg(v) = m$

connected graph without cycles – tree

graph without cycles – forest

spaning tree – spanning subgraph = tree

dense: $m = O(n^2)$

sparse: $m = O(n)$

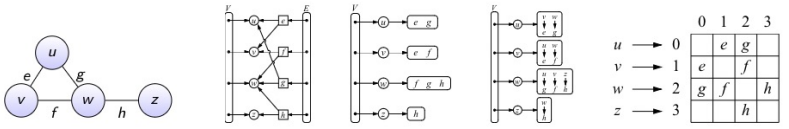
connected – path between every vertex

tree – connected undirected graph

forest – undirected graph without cycles

If G is a tree then:	If G is a forest then:	If G is connected then:
(a) $m = n$	(a) $m = n$	(a) $m = n$
(b) $m > n-1$	(b) $m > n-1$	(b) $m > n-1$
(c) $m \ge n-1$	(c) $m \ge n-1$	(c) $m \ge n-1$
(d) $m = n-1$	(d) $m = n-1$	(d) $m = n-1$
(e) $m \le n-1$	(e) $m \le n-1$	(e) $m \le n-1$

	Edge list	Adj. list	Adj. map	Adj. matrix
Space usage	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n+m)$	$\mathcal{O}(n^2)$
`outgoingEdges(v)`	$\mathcal{O}(m)$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(n)$
`getEdge(v,w)`	$\mathcal{O}(m)$	$\mathcal{O}(\min(\deg(v), \deg(w)))$	$\mathcal{O}(1)$ exp.	$\mathcal{O}(1)$
`insertVertex(x)`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(n^2)$
`insertEdge(v,w,x)`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$ exp.	$\mathcal{O}(1)$
`removeVertex(v)`	$\mathcal{O}(m)$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(\deg(v))$	$\mathcal{O}(n^2)$
`removeEdge(e)`	$\mathcal{O}(1)$	$\mathcal{O}(1)$	$\mathcal{O}(1)$ exp.	$\mathcal{O}(1)$

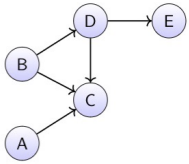


Lecture 8.2.

How many of the sequences below are a topological ordering of the graph?

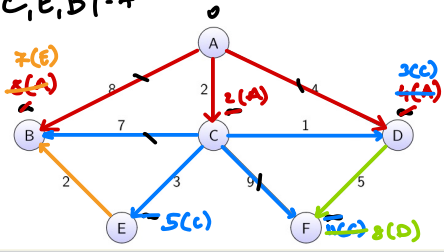
A<C
B<D<C
D<E

(a) A, B, D, C, E
(b) A, B, C, D, E
(c) B, D, E, A, C
(d) B, C, E, A, D



Use Dijkstra's algorithm to compute the shortest path between A and B

{A, C, E, B} = 7



Perform Kruskal's algorithm