

Name:

ID Number:

Exam Type:

...B...

Exam Number:

.....

Grade: 1: ... 2: ... 3: ... 4: ... 5: ... 6: ... 7: ... 8: ... 9: ... 10: ... Total:

SOLVE all the problems IN THE SPACE PROVIDED

THERE ARE 6 (SIX) PAGES THIS PAGE INCLUDED
Read and Sign the statement below at the end of the exam
Unsigned exams will be marked with grade 0 (zero).

STATEMENT

On my honor, I pledge that I have not violated the provision of the NJIT Student Honor Code.

Sign below at the end of the exam

Signature

.....

List of useful formulae

$$\sum_{i=1}^n i = \frac{n(n+1)}{2}, \quad \sum_{i=1}^n i^2 = \frac{n(n+1)(2n+1)}{6}, \quad \sum_{i=1}^n i^3 = \frac{n^2(n+1)^2}{4} \quad n! \approx \left(\frac{n}{e}\right)^n, \quad a^{\log_b n} = n^{\log_b a},$$

For $x \neq 1$, we have that

$$\sum_{i=0}^n x^i = \frac{x^{n+1} - 1}{x - 1}, \quad \sum_{i=0}^{n-1} ix^i = \frac{(n-1)x^{n+1} - nx^n + x}{(1-x)^2}, \quad \sum_{i=0}^{\infty} i \cdot (1/2)^i = 2$$

NOTE: Give an algorithm means, present the algorithm, show that it works as claimed, and analyze its running time by providing an asymptotically tight bound

Master Method. $T(n) = aT(n/b) + f(n)$, such that $a > 0$, $b > 1$, $f(n) > 0$.

M1 If $f(n) = O(n^{\lg_b a - \epsilon})$ for some constant $\epsilon > 0$, then $T(n) = \Theta(n^{\lg_b a})$.

M2 If $f(n) = \Theta(n^{\lg_b a} \lg^k n)$, then $T(n) = \Theta(n^{\lg_b a} \lg^{k+1} n)$, where $k \geq 0$ is a non-negative constant.

M3 If $f(n) = \Omega(n^{\lg_b a + \epsilon})$ for some constant $\epsilon > 0$, and if $af(n/b) \leq cf(n)$ for some constant $c < 1$ and for large n , then $T(n) = \Theta(f(n))$.

Problem 1. (6 POINTS)

Answer with a **TRUE** or a **FALSE** or give a brief answer.

- (a) Did you write your ID on page 1?
- (b) Name 3 divide-and-conquer algorithms introduced in class. (3 points)
- (c) Name a greedy algorithm introduced in class. (2 points)

Problem 2. (30 POINTS)

Give a **TRUE** or **FALSE** for each one of the statements below. Any answer other than a full **TRUE** or **FALSE** will be considered wrong. All algorithms are the book/notes implementations.

- (1) MergeSort sorts in-place.
- (2) HeapSort is stable.
- (3) On the sorted sequence $\langle 1, 2, \dots, n \rangle$, MergeSort is asymptotically faster than QuickSort.
- (4) On the reverse-sorted sequence $\langle n, n-1, \dots, 1 \rangle$, InsertionSort is asymptotically faster than QuickSort.
- (5) The asymptotic solution of $T(n) = T(n/2) + n$ is $T(n) = \Theta(n)$.
- (6) The running time of QuickSort is $O(n \lg n)$.

Problem 3. (30 POINTS)

Solve the following recurrence relations by providing asymptotically tight bounds. You only need to provide the bound, intermediate derivations are not required. If no boundary case is given, the choice of the constants is yours. You may assume that $T(n)$ is positive and monotonically increasing, if you need to do so.

- (1) $T(n) = 4T(n/2) + n^2.$
- (2) $T(n) = 8T(n/2) + n$
- (3) $T(n) = T(n-1) + n/2.$

Problem 4. (35 POINTS)

For this problem input size is specified separately for each case and thus **Row-1,Column-1** indicates a problem size of n^2 keys. Graphs have n vertices and m edges and represented using an adjacency matrix. Find the worst-case running time of every (efficient) algorithm in **Column-1** by using the time bounds in **Column-2**; use only class-introduced algorithms. If multiple answers are possible, use the asymptotically tightest bound. The same answer in **Column-2** can be reused. Row 7 is filled for you; you should fill the remaining rows. If you see $\lg^t n$, it means $(\lg n)^t$.

Column-1	Answer	Column-2
1: HeapSort of n^2 keys		A. $O(\lg n)$
2: BuildMinHeap of $n \lg n$ keys		B. $O(n \lg n)$
3: Find the number of edges of a graph		C. $O(n^2)$
4: InsertionSort on $n \lg n$ keys		D. $O(n \lg^2 n)$
5: Linear Search on n^2 keys		E. $O(n^2 \lg n)$
6: QuickSort of $n \lg n$ keys		F. $O(n)$
7: MergeSort of n keys	B	G. $O(n^2 \lg^2 n)$.
8: MergeSort of $n \lg n$ keys		H. $O(n^3 \lg^2 n)$

Problem 5. (36 POINTS)

Solve exactly the following recurrence. You may assume that n is a power of two.

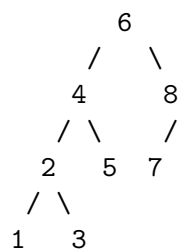
$$T(n) = 2T(n/2) + 6n, \quad T(2) = 6.$$

Problem 6. (30 POINTS)

A set A of n integers is given and also integer t . Determine in worst case time $o(n^2)$ (little oh) whether there exist two different integers x, y of A so that $x + y - t = 0$. (You may assume all integers are distinct. 'Determine' means you print ONE pair x, y if it exists or 'NOTFOUND' if no such pair exists.)

Problem 7. (60 POINTS)

(a) Perform inorder, preorder and postorder traversals on the following tree.



(b) Compress the 20-character long text `DATUMDATUMDATUMDATAD` using Huffman coding. How many bits to represent the encoded string only?

Problem 8. (46 POINTS)

Fill in the table below by providing the running time for every one of the indicated operations and every one indicated data structure of **size** n . Express the answer in terms of n . Fill entries by writing n to denote running time of $O(n)$ or $\Theta(n)$, 1 to denote running time of $\Theta(1)$, and so on. All keys already in data structure are distinct. We fill in the entry for Succ(z) into an SDL. Since it is $\Theta(1)$ we write 1.

- (a) **Ins** is Insertion of a key x into data structure,
- (b) **Del** is Deletion of a record of the data structure pointed by a pointer/reference (for linked structures) or index (for array-based structures) z , i.e. the location that will be deleted is known,
- (c) **Sea** is Search for key x .
- (d) **FindMin** is FindMinimum, i.e. determine and return the value of the key with the smallest value among the keys of the data structure; the key remains in it.
- (g) **ExMax** is ExtractMax, that is return and remove the key with largest value.
- (h) **Succ** is the successor of z , i.e. the smallest among all the elements greater than or equal to the value pointed by pointer/reference z (it has nothing to do with pointers/references).

For removal operations the data structures must be maintained in likewise form after the operation (i.e. there are no gaps or locations with no keys).

SA Sorted Array (SA) with elements ordered in increasing key order.

RSL ReverseSorted Singly Linked List (RSL) with the elements in decreasing order and the largest value key pointed by a head pointer; no tail pointer.

SDL Sorted Doubly Linked List (SDL) with the elements in increasing order and the smallest value key pointed by a head pointer; no tail pointer.

MXH A max-heap stored as an array $A[0..heapsize(A) - 1]$, where $heapsize(A) = n < length(A)$, i.e. A is big enough for a new key to be inserted. $A[0]$ is holding the MAX key.

MNH A min-heap similarly (and symetrically) defined to the MXH definition.

	SA	RSL	SDL	MNH
Ins(x)				
Del(z)				
Sea(x)				
FindMin				
ExMax				
Succ(z)			1	

Problem 9. (30 POINTS)

You are given two sorted sequences stored in arrays X and Y of n and n^2 keys respectively. We are interested in forming the union $Z = X \cup Y$, i.e. an array Z containing all keys of X and Y but duplicates are included only once. Give a space and time efficient algorithm for solving this problem. (Time efficient means its running time in terms of key comparisons cannot be asymptotically improved upon.) Justify your arguments.

Problem 10. (30 POINTS)

(Time efficient means asymptotic time efficiency cannot be improved upon. Space efficiency is to be interpreted likewise.)

You are given n^2 keys in array B . The keys have no particular order. We would like to determine the n largest of the n^2 keys and display them in the output in reverse sorted order (larger to smaller). Give a space and time efficient algorithm, explain the algorithm, analyze its worst-case running time and space complexity. Justify your arguments.