INFO 6205 – Program Structures and Algorithms
Crash Course in Algorithms – Quiz Questions

Student Name: Vidhi Bharat Patel
Professor: Nik Bear Brown

Heap Sort Algorithm

Question 1

Suppose you have an array 'arr' of size 'n', where the first 'n-k' elements are already sorted in ascending order, and the remaining 'k' elements are in arbitrary order. What is the best-case time complexity of using heapsort algorithm to sort the entire array?

A. O(n log n)
B. O(n log k)
C. O(k log n)
D. O(k^2)

Solution:

The best-case scenario for heapsort is when the input array is already sorted. In this case, building the binary heap can be done in linear time since the heap is already complete and no sifting is needed.
Therefore, the best-case time complexity of using heapsort algorithm to sort the entire array would be:

- Building the heap: O(n)
- Extracting elements from the heap and placing them in the sorted array: O(n log n)

Overall, the best-case time complexity is O(n log n), which is the same as the worst-case time complexity.

Therefore, the correct answer is A. O(n log n).

Question 2

Suppose you are given an array 'arr' of size n, where the elements are integers in the range [1, k], and k is much smaller than n. You are asked to sort the array using heapsort algorithm. Which of the following statements about this problem is true?

A. The space complexity of heapsort algorithm for this problem is O(n).
B. The best-case time complexity of heapsort algorithm for this problem is O(n log n).
C. The worst-case time complexity of heapsort algorithm for this problem is O(n log n).
D. The in-place version of heapsort algorithm can be used to sort the array.

Solution:

The worst-case time complexity of heapsort algorithm for any input array of size n is O(n log n), regardless of the range of values in the array. Therefore, option C is true.

The best-case time complexity of heapsort algorithm is O(n) when the input array is already sorted. However, in this problem, the range of values in the array is much smaller than n, which means that the input array is likely to be unsorted. Therefore, option B is false.

The space complexity of heapsort algorithm is O(1) for the standard version and O(n) for the recursive version. In this problem, we are using the standard version of heapsort, which means that the space complexity is O(1). Therefore, option A is false.

The in-place version of heapsort algorithm can be used to sort the array without using additional memory, which means that the space complexity is O(1). However, the in-place version of heapsort algorithm requires more complex implementation compared to the standard version. Therefore, we are using the standard version of heapsort algorithm in this problem. Therefore, option D is false.

Therefore, the correct answer is option C. The worst-case time complexity of heapsort algorithm for this problem is O(n log n).

Question 3

Bob and Alice are working on a project that involves processing large amounts of customer data. They need to sort the data by customer ID, which is a unique integer value assigned to each customer. Which of the following statements about the application of heapsort algorithm in this project is true?

A. The stability of heapsort algorithm is a major concern in this project.
B. The space complexity of heapsort algorithm is a major concern in this project.
C. The worst-case time complexity of heapsort algorithm is a major concern in this project.
D. The in-place version of heapsort algorithm is preferred in this project.

Solution:

The customer data consists of a large amount of records, and the customer ID is a unique integer value. Therefore, there are no equal elements, and the stability of the sorting algorithm is not a major concern. Therefore, option A is false.

The space complexity of heapsort algorithm is O(1) for the standard version and O(n) for the recursive version. In this project, we are dealing with a large amount of data, which means that the space complexity is a major concern. Therefore, option A is true.

The worst-case time complexity of heapsort algorithm is O(n log n), which is efficient for large datasets. Therefore, the time complexity is not a major concern in this project. Therefore, option C is false.

The in-place version of heapsort algorithm can be used to sort the data without using additional memory, which means that the space complexity is O(1). However, the in-place version of heapsort algorithm requires more complex implementation compared to the standard version, and it may not be necessary in this project. Therefore, option D is false.

Therefore, the correct answer is option B. The space complexity of heapsort algorithm is a major concern in this project.

Question 4

Which of the following statements about heapsort algorithm are true?

A. Heapsort is an in-place sorting algorithm.
B. Heapsort is a stable sorting algorithm.
C. Heapsort has a worst-case time complexity of O(n^2).
D. Heapsort works by first building a binary search tree.
E. Heapsort works by first building a heap.

Select all that apply.

Solution:

A. Heapsort is an in-place sorting algorithm, meaning it uses only a small constant amount of extra memory beyond the original array. This statement is true.

B. Heapsort is not a stable sorting algorithm, because it does not preserve the relative order of equal elements. This statement is false.

C. Heapsort has a worst-case time complexity of O(n log n), which is better than O(n^2) for comparison-based sorting algorithms. This statement is false.

D. Heapsort does not work by first building a binary search tree, but instead by building a heap, which is a binary tree that satisfies the heap property. This statement is false.

E. Heapsort works by first building a heap, which is a binary tree that satisfies the heap property. This statement is true.

F. Therefore, the correct statements are A) and E).

Question 5

Which of the following is a correct description of the heapify operation used in the heapsort algorithm?

A. Starting from the root node, compare the value of each node with its children and swap it with the largest (or smallest) child until the heap property is satisfied.
B. Starting from the last non-leaf node, compare the value of each node with its children and swap it with the largest (or smallest) child until the heap property is satisfied.
C. Starting from the root node, compare the value of each node with its parent and swap it with the larger (or smaller) parent until the heap property is satisfied.
D. Starting from the last non-leaf node, compare the value of each node with its parent and swap it with the larger (or smaller) parent until the heap property is satisfied.

Solution:

A. This is not a correct description of the heapify operation, because it only considers the root node and its children, while ignoring the rest of the tree. This statement is false.

B. This is a correct description of the heapify operation, because it starts from the last non-leaf node (which is the parent of the last element in the array) and compares it with its children, then moves up the tree until the root node is reached. This statement is true.

C. This is not a correct description of the heapify operation, because it swaps nodes with their parents, which violates the heap property. This statement is false.

D. This is not a correct description of the heapify operation, because it starts from the last non-leaf node and compares it with its parent, which is the opposite of the correct direction. This statement is false.

E. Therefore, the correct statement is B) Starting from the last non-leaf node, compare the value of each node with its children and swap it with the largest (or smallest) child until the heap property is satisfied.

Question 6

What is an efficient algorithm to find the k-th smallest element in a binary tree with n nodes, and what is its time complexity?

A. Perform an in-order traversal of the binary tree, and return the k-th element encountered. Time complexity: O(n).
B. Use the heapsort algorithm to sort the binary tree in ascending order, and return the k-th element. Time complexity: O(n log n).
C. Build a min-heap of the first k nodes in the binary tree, and iterate over the remaining n-k nodes. For each node, if its key value is larger than the root of the min-heap, skip it. Otherwise, remove the root of the min-heap and insert the current node into the heap. After iterating over all n nodes, return the root of the min-heap. Time complexity: O(n log k).
D. Build a max-heap of the first k nodes in the binary tree, and iterate over the remaining n-k nodes. For each node, if its key value is smaller than the root of the max-heap, skip it. Otherwise, remove the root of the max-heap and insert the current node into the heap. After iterating over all n nodes, return the root of the max-heap. Time complexity: O(n log k).

Solution:

The efficient algorithm to find the k-th smallest element in a binary tree with n nodes is C) Build a min-heap of the first k nodes in the binary tree, and iterate over the remaining n-k nodes. For each node, if its key value is larger than the root of the min-heap, skip it. Otherwise, remove the root of the min-heap and insert the current node into the heap. After iterating over all n nodes, return the root of the min-heap. The time complexity of this algorithm is O(n log k).

Option A) Perform an in-order traversal of the binary tree and return the k-th element encountered is not an efficient algorithm as it requires traversing the entire tree and has a time complexity of O(n).

Option B) Use the heapsort algorithm to sort the binary tree in ascending order, and return the k-th element is not an efficient algorithm either. The time complexity of heapsort algorithm is O(n log n) and sorting the entire tree to find the k-th smallest element would be wasteful.

Option D) Build a max-heap and iterate over the remaining nodes is not a correct algorithm as it would give the k-th largest element and not the k-th smallest element. The correct algorithm requires using a min-heap.

Question 7

Which of the following is not a valid application of heapsort algorithm?

A. Sorting an array of integers.
B. Sorting an array of floating-point numbers.
C. Sorting a linked list.
D. Finding the median of an array of integers.

Solution:

The answer is C) Sorting a linked list. The heapsort algorithm is not suitable for sorting a linked list because it requires random access to elements, which is not possible in a linked list. The heapsort algorithm involves swapping elements within an array based on their indices, which cannot be done efficiently in a linked list.

Options A) Sorting an array of integers and B) Sorting an array of floating-point numbers are both valid applications of the heapsort algorithm. The time complexity of heapsort algorithm is O(n log n), which is considered efficient for sorting large arrays.

Option D) Finding the median of an array of integers is also a valid application of the heapsort algorithm. To find the median, we can first sort the array using heapsort, and then return the middle element if the array has an odd number of elements, or the average of the middle two elements if the array has an even number of elements. The time complexity of finding the median using heapsort is O(n log n).

Question 8

Consider an array of n integers that contains both positive and negative values. Which of the following statements about applying heapsort algorithm to this array is/are true?

A. The heapsort algorithm can be modified to sort the array in descending order.
B. The worst-case time complexity of heapsort algorithm for this array is O(n log n).
C. The heapsort algorithm will not work correctly if the array contains a very large number of negative values.
D. The heapsort algorithm is not suitable for parallel processing.
E. The space complexity of the heapsort algorithm is O(1).

Solution:

A) True. The heapsort algorithm can be modified to sort the array in descending order by simply reversing the comparison operator used to build the heap. Instead of building a max-heap, we can build a min-heap to sort the array in descending order.

B) True. The worst-case time complexity of heapsort algorithm is O(n log n), regardless of the values in the array. Although the presence of negative values in the array may affect the performance of some comparison-based sorting algorithms, heapsort is not affected by this.

C) False. The heapsort algorithm works correctly on arrays with positive and negative values. The presence of negative values may affect the ordering of elements in the heap, but this does not affect the correctness of the algorithm.

D) False. The heapsort algorithm is suitable for parallel processing because it is a comparison-based sorting algorithm that can be easily parallelized using techniques such as parallel heap construction and parallel heap sort.

E) False. The space complexity of the heapsort algorithm is O(1) for in-place heapsort, but O(n) for out-of-place heapsort. In in-place heapsort, the input array is used to store the heap, so no extra space is required. However, in out-of-place heapsort, a separate array is used to store the heap, so the space complexity is O(n).

Question 9

Consider a scenario where we want to sort an array of n integers, but we have limited memory available. Which of the following statements about using heapsort algorithm in this scenario is/are true?

A. We can use an external sorting technique such as merge sort to sort the array.
B. We can use an in-place heapsort algorithm to sort the array.
C. We can use a variation of heapsort called tournament sort to sort the array.
D. We can use a distributed sorting algorithm such as MapReduce to sort the array.
E. We can use a parallel sorting algorithm such as quicksort to sort the array.

Solution:

A) True. External sorting techniques such as merge sort are suitable for sorting large datasets that cannot fit into memory. The idea is to divide the dataset into smaller chunks that can fit into memory, sort each chunk separately, and then merge the sorted chunks into a single sorted dataset.

B) False. In-place heapsort algorithm requires O(1) extra memory, but still requires O(n) space for the input array itself. Therefore, in this scenario where we have limited memory available, in-place heapsort may not be a feasible option.

C) True. Tournament sort is a variation of heapsort that uses a tournament-style approach to construct the heap. It is a comparison-based sorting algorithm that requires O(n) extra memory, and is suitable for scenarios where the input data is large but can still fit into memory.

D) True. Distributed sorting algorithms such as MapReduce are suitable for sorting very large datasets that are distributed across multiple machines. The idea is to divide the dataset into smaller chunks that can be processed in parallel on different machines, and then combine the sorted chunks into a single sorted dataset.

E) True. Parallel sorting algorithms such as quicksort are suitable for scenarios where the input data is large and can be divided into smaller chunks that can be processed in parallel on different threads or machines. The idea is to divide the input array into smaller sub-arrays, sort each sub-array independently in parallel, and then merge the sorted sub-arrays into a single sorted array.

Question 10

Consider a scenario where we want to sort a linked list of n elements using heapsort algorithm. Which of the following statements about this scenario is/are true?

A. Heapsort algorithm can be used to sort a linked list in O(nlogn) time.
B. Heapsort algorithm requires O(1) extra memory for sorting a linked list.
C. Heapsort algorithm can be used to sort a linked list in-place.
D. Heapsort algorithm requires the linked list to be converted to an array before sorting.
E. Heapsort algorithm is not suitable for sorting a linked list.

Solution:

A) False. Heapsort algorithm requires random access to elements in the input array, which is not possible in a linked list. Therefore, heapsort algorithm cannot be used to sort a linked list in O(nlogn) time.

B) False. Heapsort algorithm requires extra memory for constructing the heap, which is not possible in a linked list. Therefore, heapsort algorithm cannot be used to sort a linked list with O(1) extra memory.

C) False. In-place heapsort algorithm requires O(1) extra memory for sorting an array, but it is not possible to sort a linked list in-place using heapsort algorithm. Therefore, heapsort algorithm cannot be used to sort a linked list in-place.

D) False. Heapsort algorithm requires random access to elements in the input array, which is not possible in a linked list. Therefore, the linked list cannot be directly used with heapsort algorithm, and it is not necessary to convert the linked list to an array before sorting.

E) False. Heapsort algorithm can be adapted to sort a linked list by using a variation called "heapsort for linked list". The idea is to use a min-heap or max-heap to repeatedly extract the minimum or maximum element from the linked list, and then add it to the sorted portion of the list. This process continues until the entire list is sorted. However, this variation has a time complexity of O(n^2), which is worse than the time complexity of O(nlogn) for heapsort algorithm on an array.