Analysis of Algorithms

BLG 335E

Project 1 Report

Mehmet Ali Balıkçı
balikci20@itu.edu.tr

Faculty of Computer and Informatics Engineering
Department of Computer Engineering
Date of submission: 14.12.2023

1. Implementation

1.1. Implementation of Quick Sort with Different Pivoting Strategies

The project incorporates a total of 12 main functions and 5 utility functions. It's worth noting that the space complexity for these functions is O(1), signifying that they utilize a constant amount of space irrespective of input size. This ensures efficient memory usage in various scenarios.
The main functions and their mission are explained below.

Main Functions:

- max_heapify

Takes the data vector and, index of the node, the heap size, and number of children of non-leaf node if it is different from 2. It finds the indexes of children of a node and compare this node with its children. If its children is bigger than it, then the exchange operation is done. It is capable of working with any d-ary heap. Recurrence relation is T(n) <= T(2n/3) + O(1), and if it is solved by master theorem then the complexity will be T(n) = O(logn).

- build_max_heap

Takes the vector containing data from the file and the degree of the heap as arguments. If used with a d-ary heap, the degree must be passed as an argument; otherwise, the default value is set to 2. The function builds a max heap, ensuring that each parent node is greater than its children nodes. The MaxHeapify function is called approximately n/d times, where 'n' is the number of elements in the vector and 'd' is the degree of the heap. If d is 2, this occurs n/2 times. Since the complexity of MaxHeapify is O(logn), the overall complexity of BuildMaxHeap is n/2 * logn, resulting in O(nlogn). Through simplification, a tighter bound of O(n) can be achieved.

- heapsort

Takes the vector containing the data from the file as an argument. Using BuildMaxHeap and MaxHeapify functions, it arranges the vector in ascending order, creating a sorted heap. This sorting algorithm is applicable to any d-ary heap, as BuildMaxHeap and MaxHeapify functions are designed to work seamlessly with different heap degrees. The time complexity is approximately O(n) for BuildMaxHeap, and each call to MaxHeapify takes O(logn) time. Therefore, the overall time complexity is O(n) + O((n-1) * logn) = O(nlogn).

- heap_increase_key

Takes the data vector, the index of the node whose value will be increased, and the new value for this node as arguments. The function assigns the new value to the specified node and adjusts its position by swapping it with its parent node until it becomes greater than its child. The time complexity is O(logn) since the function involves a single loop that iterates up the tree at most to the depth of log(n), where ’n’ is the number of elements in the heap.

- max_heap_insert

Takes the data vector, a key value (in this data set, a city name), and the value of the node to be inserted as arguments. The function adds a node to the end of the heap, assigns this node the value of minus infinity, and then calls the HeapIncreaseKey function to maintain the max-heap property. The time complexity of MaxHeapInsert is O(logn). Apart from the assignment operations, there is a single call to the HeapIncreaseKey function, whose complexity is O(logn).

- heap_maximum

Takes the data vector as an argument and returns the maximum element of the heap, which is positioned as the first element in the max-heap arrangement. This function is designed to work seamlessly with any d-ary heap. The time complexity of HeapMaximum is O(1), as it involves only a return operation.

- heap_extract_max

Takes the data vector as an argument and returns the maximum element of the heap, which is positioned as the first element in the max-heap arrangement. This function is designed to work seamlessly with any d-ary heap. The time complexity of HeapMaximum is O(1), as it involves only a return operation.

- dary_calculate_height

Takes the number of nodes and d, which is the number of children of non-leaf nodes. The time complexity is O(1), as it involves only a calculation operation.

- dary_extract_max

Takes the data vector as an argument. It swaps the first element, which is the maximum element due to the heap being a d-ary max-heap, with the last element and removes the last element from the heap. Then, it calls the MaxHeapify function for the first element to preserve the max-heap structure. Finally, it returns the maximum element that was removed from the heap. The function is designed to work with any d-ary heap. The time complexity is proportional to the complexity of MaxHeapify, which is typically expressed as O(d * (logn / logd), considering the tree is d-ary.

- dary_insert_element

Takes the data vector, a key value (a city name in this data set), and the value of the node to be inserted as arguments. The function adds a node to the end of the heap, assigns this node the value of minus infinity, and then calls the DaryIncreaseKey function to maintain the max-heap property. The function is designed to work with any d-ary heap. The time complexity is proportional to the complexity of DaryIncreaseKey, typically expressed as O(d * (logn / logd), considering the tree is d-ary.

- dary_increase_key

Takes the data vector, the index of the node whose value will be increased, and the new value for this node as arguments. The function assigns a new value to the specified node and adjusts its position by swapping it with its parent node until it becomes greater than its child. The function is designed to work with any d-ary heap. The time complexity is proportional to the depth of the tree, which is approximately (log(n) + logd) / logd.

- dary_heapsort

Takes the vector containing the data from the file as an argument. Using BuildMaxHeap and MaxHeapify functions, it arranges the vector in ascending order, creating a sorted heap. This sorting algorithm is applicable to any d-ary heap, as BuildMaxHeap and MaxHeapify functions are designed to work seamlessly with different heap degrees. The time complexity is approximately O(n) for BuildMaxHeap, and each call to MaxHeapify takes (log(n) + logd) / logd = O(logn) time. Therefore, the overall time complexity is O(n) + O((n-1) * [(log(n) + logd) / logd)]) = O(nlogn).

The utility functions and their mission are explained below.

Utility Functions:

- FindParent

It finds and returns the index of the parent of a node.

- Exchange

It swaps two nodes with each other.

- PrintHeaptoFile

It gets the final data or output and writes it output.csv file.

- ReadCsv

It reads the data from the file.

1.2. Differences between Quicksort and Heapsort

As seen in the table, if quicksort is implemented with either random element or median-of-three-element pivot strategies, it will be approximately ten times faster than the heapsort algorithm regardless of the way of sorting the data.

In heapsort algorithm, there are more comparisons than quicksort algorithm. More comparisons will cause the algorithm to become more complex and thus increase its complexity.

In general, on average, quicksort tends to have fewer comparisons than heapsort. Quicksort often outperforms heapsort in practice, especially when implemented with a good pivot strategy. However, in the worst-case scenario, quicksort can have more comparisons than heapsort.

Heapsort, on the other hand, guarantees a worst-case time complexity of O(n log n) for comparisons. It may have a slightly higher constant factor compared to quicksort in practice, but its worst-case performance is consistent.

In summary, if looking for a sorting algorithm with a guaranteed worst-case time complexity, heapsort may be preferable. If average-case performance is more important, quicksort with a good pivot strategy is often a strong choice. The exact comparison count can vary based on specific circumstances and implementations.

	Population1	**Population2**	**Population3**	**Population4**
Heapsort	0.019623	0.024275	0.020143	0.024012
QS– Last element	0.105638	1.877420	0.806743	0.006975
QS– Random element	0.007248	0.002326	0.008487	0.007007
QS– Median of three elements	0.007247	0.002114	0.009217	0.006994

Table 1.1: Comparison of heapsort and quicksort implemented with different pivot strategies

1.3. Appendix

- Using Vectors in the Project

I used vectors in my project because the data we wanted to use had to be kept in pairs and data extraction and data addition operations had to be done easily on this data. Vectors were the most suitable data structure for this, so I used vectors.

- Calling functions

build_max_heap, heapsort, heap_maximum, heap_extract_max, dary_extract_max, dary_calculate_height You don’t need to enter any arguments for the above functions. It will work when you type the name of the functions. The others:

max_heapify Function:

./Heapsort population2.csv max_heapify out.csv i11

If you write it like this, the function will work. If you enter any letter other than ’i’, the function will return an error message.

heap_increase_key Function:

./Heapsort population2.csv heap_increase_key out.csv i11 k23

Here I designed the function to work regardless of the order in which you enter ’i’ and ’k’. You can operate it by changing their locations. If you enter a letter other than ’i’ and ’k’, the function will return an error message.

max_heap_insert Function:

./Heapsort population2.csv max_heap_insert out.csv k_konya_42

You can run it as specified in the report. If you enter the first letter ’k’ incorrectly, the function will give an error, it is designed this way.

dary_increase_key Function:

./Heapsort population2.csv dary_increase_key out.csv i11 k23 d3

Here I designed the function to work no matter what order you enter ’i’, ’k’ and d’. You can operate it by changing their locations. ’l’ If you enter a letter other than , ’k’ and ’d’, the function will return an error message.

dary_insert_element Function:

./Heapsort population2.csv dary_insert_element out.csv k_konya_42 d3

You can write a command like above to run this function. Here, the function will work regardless of whether k or d comes first. It is designed this way. If you enter a letter other than ’k’ or ’d’, it will give an error.

dary_heapsort Function:

./Heapsort population2.csv dary_heapsort out.csv d3

You can run this function by typing a command like above on the command line. If you enter a character other than ’d’, the function will give an error, or if you do not give any parameters, the function will give an error again.