

Laboratory Session 04

Sorting Pt. 2

1 Priority Queue (Max-Heap)

In this exercise, you must implement a Priority Queue using a **Max-Heap** data structure based on an array.

You must implement the following three functions. The heap must be maintained *in-place* within the provided list *A*.

1. `build_max_heap(A, heap_size)`: Takes an unordered list *A* of integers and converts it into a valid Max-Heap using a bottom-up approach.
2. `extract_max(A, heap_size)`: Removes and returns the maximum element (the root) from the heap *A*. It must then restore the Max-Heap property for the remaining elements. Return `None` if the heap is empty.
3. `search(A, value)`: Searches for a specific `value` inside the heap *A* and returns its index. Return `-1` if the value is not found. Try to optimize the search by avoiding branches where elements are strictly smaller than the target value.

State the asymptotic time complexity for each of the three functions as comments in your code.

Examples

Given $A = [4, 1, 3, 2, 16, 9, 10, 14, 8, 7]$ for the first example:

Input	Expected Output
<code>build_max_heap(A)</code>	<code>[16, 14, 10, 8, 7, 9, 3, 2, 4, 1]</code>
<code>extract_max(A)</code> <i>after the previous build</i>	16 (and <i>A</i> becomes <code>[14, 8, 10, 4, 7, 9, 3, 2, 1]</code>)
<code>search(A, 9)</code> <i>after the previous extraction</i>	5
<code>search(A, 100)</code>	-1

2 String Radix Sort (LSD)

Write a function `string_radix_sort(A)` that sorts a list *A* of lowercase English strings in **alphabetical order** using the Radix Sort algorithm (LSD - Least Significant Digit approach).

Since strings may have **variable lengths**, your algorithm must process characters starting from the rightmost position of the longest string. Treat missing characters in shorter strings as conceptually preceding the letter 'a' (e.g., using a padding or a specific "empty" bucket).

State the corresponding asymptotic time complexity at the end of the procedure.

Examples

Input	Expected Output
<code>string_radix_sort(["cat", "car", "apple", "bat"])</code>	<code>["apple", "bat", "car", "cat"]</code>
<code>string_radix_sort(["a", "ab", "abc", "b"])</code>	<code>["a", "ab", "abc", "b"]</code>
<code>string_radix_sort(["zebra", "ant"])</code>	<code>["ant", "zebra"]</code>
<code>string_radix_sort([])</code>	<code>[]</code>

3 Instructions

How to Annotate Your Code

To report the asymptotic computational complexity, place a Python comment containing a $\text{\LaTeX}$ formula at the end of each function, after its final return statement. Keep the comment indented inside the function. For example, if your analysis establishes a linear upper bound, write:

```
def example(A):  
    # ... your implementation ...  
    return result  
    # Time complexity (worst case):  $O(n)$ 
```

The dollar signs and backslash commands are literal text in the Python comment. For other bounds, use valid $\text{\LaTeX}$ such as $O(n^2)$ or $O(\log n)$. Include the tight bound and your derivation in the comments immediately before the final complexity comment. Annotate any helper functions too, specifying their input-size parameter.

How to Complete This Lab

In this lab, you are provided with this PDF containing the exercise instructions, the necessary libraries (if there are libraries to use) and expected Input/Output examples (e.g., Input: `power(2, 3)` $\rightarrow$ Output: 8).

Along with this document, you have received two Python files:

- `exercises.py`: This file contains the function signatures with empty bodies for all exercises. For example:

```
def power(base: int, exponent: int) -> int: pass
```

Your task is to replace the `pass` keyword with your actual implementation. Do not change the function names or their signatures. You can add other functions to support your code.

- `main.py`: This file contains a suite of automated tests. Once you have written your code in `exercises.py`, run `main.py`. It will automatically import your functions, test them against various inputs, and report if your outputs are correct (True for correct result, False otherwise).