

Assignment 3

Manan Patel

September 2024

1: Design this data structure by specifying the variables (array, counter, ...) involved..

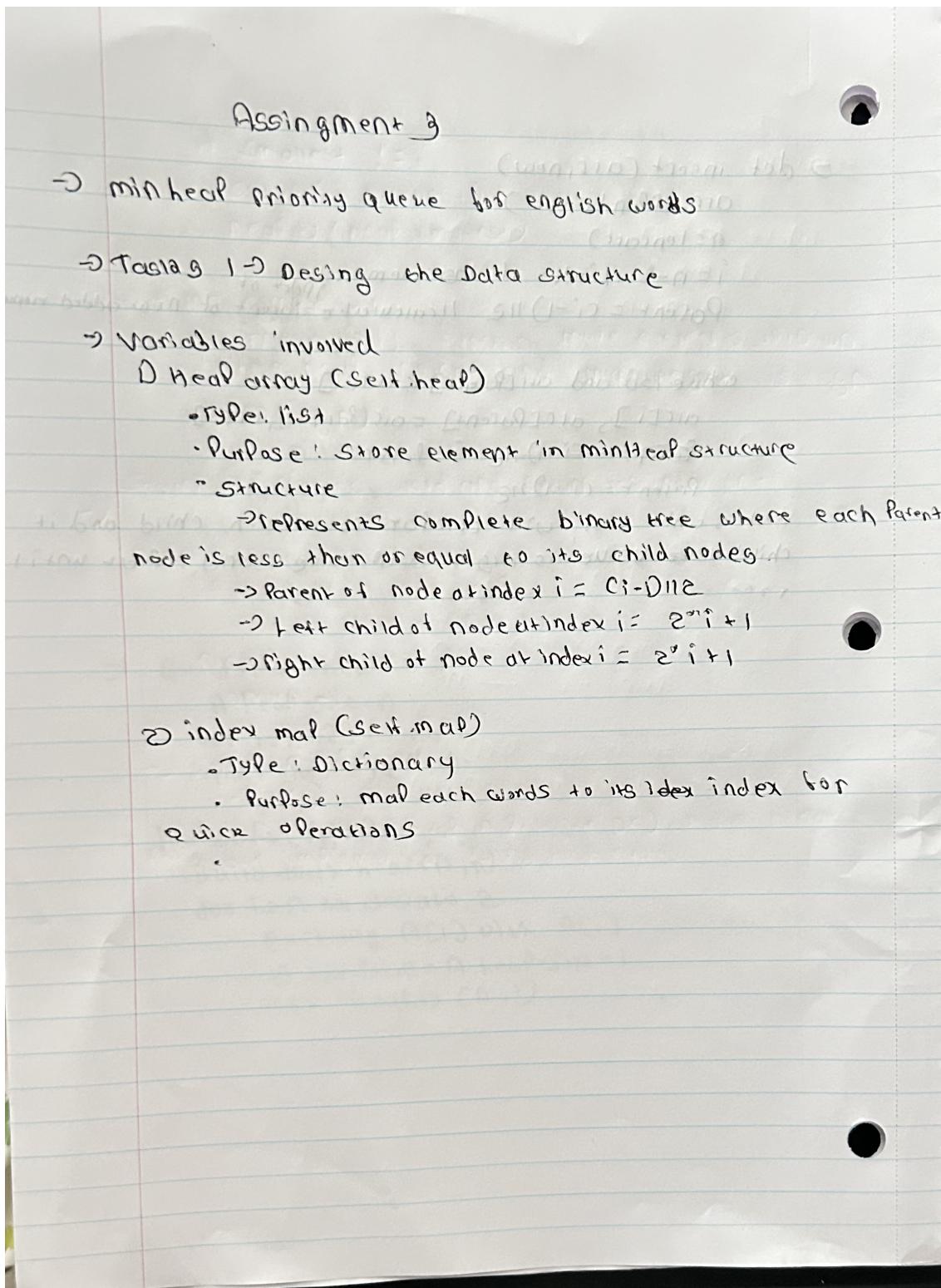


Figure 1: Answer of Task 1

2-3. Write the most efficient algorithms you can find for each of the required operation. Analyze the (worst-case) time complexity of each algorithm.

→ Task-2.3 algorithms and its time complexity

→ function `length_words()`
return `length(head)`

→ Time complexity = $O(1)$
→ retrieving length of list is a constant time operation

→ Remove first word in dictionary order if ds is not empty

function `RemoveFirstWord()`:
if `length(head) == 0`:
return "head is empty"
`first_word = head[0]`
`swap(0, -1)` // first word with last word
Remove last word from head
Delete `first_word` from index_map
`healify(head, 0)` // fix the head again
return `first_word`

function `healify(head, i)`
`smallest = i`
`left = 2*i + 1`
`right = 2*i + 1`
if `left < head.size` and `head[left] > head[smallest]`
 `smallest = left`
if `right < head.size` and `head[right] > head[smallest]`
 `smallest = right`

Figure 2: Answer of Task 2

```
if smallest != index_smallest or identical  
swap(i,smallest)  
height(thead,smallest)
```

→ time complexity

→ ~~the~~ height function move elements up down
to the head height and max height is $\log n$
→ $O(\log n)$

→ Search for specific word and return its index or -1

function search(word):

```
if word in index_map  
    return index_map[word]  
else  
    return -1
```

→ time complexity

→ $O(1)$

Dictionary lookup is O(1) because
they are implemented using hash tables

(1,900) external neurons

1110-7191

1119-1187

+ 791 8137-+2011002

+ 1818-+2011002

Figure 3: Answer of Task 2

→ add new element if it is not already in queue

```
function insert(new)
    if new in index map
        return "word is already present in queue"
    heap.append(new)
    i = index of new
    index_map[new] = i
    while i > 0 and heap[Parent(i)] > heap[i]
        swap(i, Parent(i))
        i = Parent
        Parent = ci - 1 || 2
```

```
function swap(i, j)
    exchange heap[i] with heap[j]
    index_map[heap[i]] = i
    index_map[heap[j]] = j
```

→ Time complexity

→ $O(\log n)$

→ in worst case new word will be smaller than all existing elements and needs to move to root of the heap since height of heap is $\log n$
max num of swap = $\log n$
 $\therefore O(\log n)$

Figure 4: Answer of Task 2

Operation	Worst-Case Time Complexity
Insert a new word	$O(\log n)$
Remove the first word	$O(\log n)$
Search for a specific word	$O(1)$
Report the number of words	$O(1)$

Table 1: Worst-Case Time Complexity of Each Operation

4-5: Implement the data structure with its operations. Use a program to test the correctness of the design.

Program Output

```

if __name__ == "__main__":
    pq = priorityqueue()
    pq.insert("apple")
    pq.insert("cherry")
    pq.insert("pineapple")
    pq.insert("banana")
    # Print the current state of the queue
    pq.print()
    print()
    # Report the number of words
    pq.report_nwords()
    print(pq.insert("banana"))
    # Remove the first word in dictionary order
    print(f"first word is : {pq.remove_first_word()}")
    # Print the queue after removal
    pq.print()
    print()
    # Report the number of words after removal
    pq.report_nwords()
    # Search for a specific word
    print(f"word banana is present at: {pq.search('banana')} index")
    # Search for a word not in the queue
    print(pq.search("durian"))

✓ 0.0s
Current heap: ['apple', 'banana', 'pineapple', 'cherry']

Number of words in the queue is: 4
Word banana is already present in the queue
first word is : apple
Current heap: ['banana', 'cherry', 'pineapple']

Number of words in the queue is: 3
word banana is present at: 0 index
-1

```

Figure 5: Answer of Task 2