IMPLEMENTATIONS OF METHODS AND FUNCTIONS

1. TRIE

TrieNode: This is a class which creates a node of trie. It consists of an array which points to the children nodes of that TrieNode. It contains only one function getValue() which returns the data stored in the node.Its time complexity is O(1).

Person: This is a class which defines the description of a person's data to be stored in the Trie. It has two functions: getName(); which returns the name of the person; and toString().Both the functions have time complexity O(1).

Trie: This class defines the main trie structure.It has following functions:

- Insert: This function inserts a given value according to the key associated with it. If a node contains data, then it is the end of a key.This is how a node is marked as the end of a word.The time complexity is O(h), where h= length of key.

- Search: This function searches for a given key in the trie. It returns the node if found, else it returns null. Its time complexity is also O(h),where h=length of key.
- startsWith(prefix): This function returns a TrieNode from which all the keys with this prefix can be found using its children.It has time complexity O(prefix length);
- PrintTrie(node): This function prints all values which are stored in a subtree with root node.Its worst case time complexity is O(h),where h =length of key with that node’s key as its prefix;
- Delete(key): This function deletes the node with that given key.It searches for the key and the removes it. Hence its time complexity is also O(key length).
- Printlevel(level): This function prints all the characters stored in a particular level of the tree.Its worst case time complexity is O(total no. of nodes in the trie).

- Print(): This function prints all the levels of the tree.Its worst case time complexity is also O(no. of nodes in the trie).

2.PriorityQueue:

Pair: This class creates a pair of two generic objects.

Student: This class creates a student object which stores information about the student.It has three functions:

- GetName(): return name of the student. Complexity: O(1);
- CompareTo(): This function defines how two Student objects are compared. Complexity: O(1)
- toString(): This function overrides the toString function and helps in converting a Student object to a String.

MaxHeap: This class is used to create a max heap which is used as a priority queue. Further the data is stored in a pair with an integer called priority, which is is used to implement the FIFO order in case the two keys of the data are same. It has following functions:

- insert(element): This function inserts an element in the maxheap. This function has a worst case time complexity of O(log n).
- delete(int i): This function deletes a node at index i in the arrayList created in the heap; i.e., it deletes the i th element in the heap. It has a time complexity O(h), where h is the height of the subtree with node at index i as its root.
- Max(int l): This function is used to find the index of the maximum of the too children of of the node at index i. It has time complexity O(1).
- Heapify(int l): This function is used to heapify down the node at index l; i.e.,it is used to restructure the heap after removing an element. It has time complexity O(h), where h is the height

of the subtree with node at index i as its root.

- extractMax(): This function is used to extract the maximum value from the heap, and then remove it from the heap. It has a time complexity of O(log n).
- contains(key): This function is used to check if a key is present in the heap or not. It has time complexity O(n).

3. **Red Black Tree:**

RedBlackNode: This class is used to create a node for the red black tree. It has the following functions:

- getValue(): This function returns the data stored in the node.
- getValues(): This function returns a list of all the data stored in the node.

RBTree: This class is used to create a red-black tree using the above RedBlackNode. It has the following functions:

- insert(): This function is used to insert data into the RBTree. It uses a helper function to first insert the element like a simple binary tree.Then it is rearranged using another helper function restructure() with time complexity O(log n) which restructures the tree so that the black height of the tree is maintained. It also uses the helper functions rotateleft() and rotateright() to restructure the tree. It has a time complexity of O(log n).
- search(): This function is used to search a given key in the tree. If the key is found then it returns the node corresponding to that key. It also has a worst case time complexity of O(log n).

4. Project Management:

Job: This class is used to store data about a job. It implements the Job_Report interface.It has the following functions:

- compareTo(): This method is used to compare two jobs according to their priority or FIFO. It has a time complexity of O(1).

- toString(): This method overrides the toString() function of Job. It has a time complexity of O(1).

Project: This class is used to store data about a project. It also contains a list containing all the jobs of that project.

User: This class is used to create a user object.It implements the User_Report interface. It stores data about a user.

Scheduler_Driver: This class contains the main method of the project management. It is used to read data and create the output as required. It has the following functions:

- search(): It is used to search a Job in the heap of jobs. Its time complexity is O(n).
- run_to_completion(): This function keeps executing the Jobs even when the file has ended. Its worst case time complexity is O(nlog n) where n =total number of jobs.
- Handle_project(): This function handles the projects. It enters the project in the

projects trie. Its time complexity is O(project name length).

- Handle_job(): This function is used to handle job inputs. It creates a new job and inserts it in the jobs heap as well as the project’s list of jobs. It has a worst case time complexity of O(log n) where n is the number of elements in the jobs heap.
- Handle_user(): This function is used to handle user inputs. It creates a new user and inserts it into the users RBTree. It has a time complexity of O(log n).
- Handle_query(): This function is used to handle the queries. Its worst case time complexity is O(n).
- Handle_empty_line(): This function is used to handle empty line inputs. Its worst case time complexity is O(nlog n) and average case complexity is O(nlog n).

- Handle_add(): This function increases the budget of a project and also puts back all the jobs of that project from the extracted list back to the jobs heap. Its worst case time complexity is O(n^2 +h) where n is the number of elements in the heap and h is the length of the name of the project.
- Print_stats(): This function is used to print the final stats of the schedular execution. Its worst case time complexity is O(nlog n) and its average case time complexity is O(n).
- Timed top_consumer(): This function returns the list of top<top> budget consuming users sorted in order of their budgets consumed and then in order of their last completed job's time. The time complexity of this function is O(u log u) where u is the number of users.
- Timed flush(int waittime): This fun increases the priority of those jobs whose waiting time is more than the

given waittime and then executes those jobs if the budget allows. The time complexity of this function is O (n log n) where n is the total number of jobs.

- handle_new_project(): It returns a list of all jobs of the project which have arrival time between t1 and t2. Its time complexity is O((n/m)+h) where n= total number of jobs and m = total number of projects and h= length of user name.
- handle_new_user(): It returns a list of all jobs of the user which have arrival time between t1 and t2. Its time complexity is O((n/u)+log u) where n= total number of jobs and u= total number of users.
- handle_new_projectuser(): It returns a list of all jobs of the user of a given project which have arrival time between t1 and t2. Its time complexity is O(log u+(n/m)(n/p)) where u = number of users, n= total number of jobs and p = total number of projects.

- handle_new_priority(): It returns a list of all waiting jobs with a priority greater than or equal to <Priority>. The time complexity of this function is O(x) where x= total number of waiting jobs.
- timed_handle_user(): It has the same implementation as handle_user(), only the print statements are removed, the time complexity remains the same.
- timed_handle_job(): It has the same implementation as handle_job(), only the print statements are removed, the time complexity remains the same.
- timed_handle_project(): It has the same implementation as handle_project(), only the print statements are removed, the time complexity remains the same.
- timed_run_to_completion(): It has the same implementation as handle_run_to_completion(), only the print statements are removed, the time complexity remains the same.