Data Structure for a single resource reservations

Design a data structure to do reservations of future jobs on a single machine under following constraints.

1) Every job requires exactly k time units of the machine.

2) The machine can do only one job at a time.

3) Time is part of the system. Future Jobs keep coming at different times. Reservation of a future job is done only if there is no existing reservation within k time frame (after and before)

4) Whenever a job finishes (or its reservation time plus k becomes equal to current time), it is removed from system.

Example:

```
Let time taken by a job (or k) be = 4

At time 0: Reservation request for a job at time 2 in
           future comes in, reservation is done as machine
           will be available (no conflicting reservations)
Reservations {2}

At time 3: Reservation requests at times 15, 7, 20 and 3.
           Job at 7, 15 and 20 can be reserved, but at 3
           cannot be reserved as it conflicts with a
           reserved at 2.
Reservations {2, 7, 15, 20}

At time 6: Reservation requests at times 30, 17, 35 and 45
           Jobs at 30, 35 and 45 are reserved, but at 17
           cannot be reserved as it conflicts with a reserved
           at 15.
Reservations {7, 15, 30, 35, 45}.
Note that job at 2 is removed as it must be finished by 6.
```

Let us consider different data structures for this task.

One solution is to keep all future reservations sorted in array. Time complexity of checking for conflicts can be done in O(Logn) using Binary Search, but insertions and deletions take O(n) time.

Hashing cannot be used here as the search is not exact search, but a search within k time frame.

The idea is to use Binary Search Tree to maintain set of reserved jobs. For every reservation request, insert it only when there is no conflicting reservation. While inserting job, do “within k time frame check”. If there is a k distant node on insertion path from root, then reject the reservation request, otherwise do the reservation.

```
// A BST node to store future reservations
struct node
{
    int time; // reservation time
    struct node *left, *right;
};

// A utility function to create a new BST node
struct node *newNode(int item)
{
    struct node *temp =
        (struct node *)malloc(sizeof(struct node));
    temp->time = item;
    temp->left = temp->right = NULL;
    return temp;
}

/* BST insert to process a new reservation request at
   a given time (future time).  This function does
   reservation only if there is no existing job within
   k time frame of new job  */
struct node* insert(struct node* root, int time, int k)
{
    /* If the tree is empty, return a new node */
    if (root == NULL) return newNode(time);

    // Check if this job conflicts with existing
    // reservations
    if ((time-k < root->time) && (time+k > root->time))
        return root;

    /* Otherwise, recur down the tree */
    if (time < root->time)
        root->left  = insert(root->left, time, k);
    else
        root->right = insert(root->right, time, k);

    /* return the (unchanged) node pointer */
    return root;
}
```

Deletion of job is simple BST delete operation.

A normal BST takes O(h) time for insert and delete operations. We can use self-balancing binary search trees like AVL, Red-Black, .. to do both operations in O(Log n) time.

This question is adopted from this MIT lecture.