QUESTION 1

First, create a helper function for helping implement the add_sorted_child method **Fig 1**

```
void insert_child(tree_node *head, tree_node *child){

if(head->sibling == NULL)

head->sibling = child;

tree_node temp = head;

while((temp->sibling) && (temp->sibling->payload < child->payload))

{

temp = temp->sibling;

}

tree_node next = temp->sibling;

temp->sibling = child;

child->sibling = next;

}
```

Create the method add_sorted_child that calls the helper method insert_child **Fig 2**

```
void add_sorted_child(tree_node *parent, tree_node *child){

if(parent->child == NULL){

parent->child = child;

}
else{

insert_child(parent->child, child);
}


void add_sorted_child(tree_node *parent, tree_node *child){

if(parent->child == NULL){

parent->child = child;

}
else{

insert_child(parent->child, child);

}
```

Analysis

1 . When it comes to big - O the function will take big - O (m+n) such that m and n refers to the number of nodes in the two trees

2. The result in 1 can be improved by using a recursive function, which in this case is a repetitive function which makes use of its terms to form a series of sequential terms.

3 The following are changes that would be required by 2

- Conversion of the tree that is leftmost and derivative to a binary search tree for better performance
- Removal of extra variables to reduce the memory required during run time
- Reduce the complexity of the function

Question 2

Implementation of void delete_word(trie_node *trie,char *word);full function **Fig 3**

```
void delete_word(trie_node *trie, char *word){

  int l = strlen(word);
  printf("Trying to find the word %s to delete.\n", word);

  if (l == 0){
    printf("  Okay, we found it.  Is it a word?  %s\n",
           trie->is_word ? "Yes." : "No.");

    if (trie->is_word){
      trie->is_word = 0;
      actually_delete(trie);
      printf("  Deleted.\n");
    }
    else{
      printf("  Nothing to delete.\n");
      return;
    }
  }
  else{
    char c = word[0];
    char nc = which_child(c);

    if (nc == -1){
      delete_word(trie, word + 1);
    }

    trie_node *next = trie->children[nc];

    if (next == NULL){
      printf("  Nothing to delete.\n");
      return;
    }
    delete_word(next, word + 1);
  }
}
```

Dealing with a word prefix
In the above function (**Fig 3**) in this if block we deal with a prefix such that the goal-keeper is the same with
Goalkeeper

```
    if (nc == -1){
      delete_word(trie, word + 1);
    }

    trie_node *next = trie->children[nc];
```

Gracefully failing for instance do nothing and return in the case a word is not a trie
In the full function in **Fig 3**

```
  if (next == NULL){
    printf("  Nothing to delete.\n");
    return;
  }
  delete_word(next, word + 1);
```

Question 3
Implementing mhash *mhash_new();

```
mhash *Mhash_new()
{
    Mhash *hash;

    assert(weight > 0);
    assert(size > 0);
    if (IS_POW_OF_TWO(weight)) {
        if ((unsigned)weight > 0x80000000)
            return NULL;
        weight = roundup_pow_of_two(weight);
    }
    if (!NEW0(hash))
        return NULL;
    if (!ALLOC(hash->buckets, weight * sizeof(BUCKET_T))) {
        FREE(hash);
        return NULL;
    }
    memset(hash->buckets, 0, weight * sizeof(BUCKET_T));
    hash->weight  = weight;
    hash->monsteriness = monsteriness;
    return hash;
}
```

Implementing mhash_hashvalue(char * name);

```
unsigned int *mhash_hashvalue(char *name)
{
    int index;
    NODE_T *p, *value = NULL;
    BUCKET_T *bucket;

    assert(name);
    assert(hash);
    index = MOD(tuple4_hash(name), hash->len);
    bucket = &hash->buckets[index];

    p = bucket->list;
    while (p) {
        if (tuple4_equal(name, &p->name)) {
            value = p;
            break;
        }
        p = p->next;
    }
    return (void*)value;
}
```

Implementing mhash_add(mhash *hash, monster *m);

```
void *Mhash_add(Mhash *hash, monster*m)
{
    int index;
    NODE_T *value, *p;
    BUCKET_T *bucket;

    assert(m);
    assert(hash);
    value = Mpool_alloc(hash->pool);
    if (!value) {
        return NULL;
    }
    value->m = *m;
    index = MOD(tuple4_hash(m), hash->len);
    bucket = &hash->buckets[index];

    p = bucket->list;
    value->next  = p;
    bucket->list = value;
    return (void*)value;
}
```

Implementing void mhash_remove(Mhash *hash, char *name);

```c
void mhash_remove(Mhash *hash, char *name)
{
    int weights, flag = 0;
    NODE_T **p, *q;
    BUCKET_T *bucket;

    assert(name);
    assert(hash);
    weights  = MOD(tuple4_hash(name), hash->monsteriness);
    bucket = &hash->buckets[weights];

    if (!bucket->list)
        return -1;

    p = &bucket->list;
    do {
        q = (*p)->next;
        if (tuple4_equal(&(*p)->name, name)) {
            Mpool_free(hash->pool, *p);
            *p   = q;
            flag = 1;
            break;
        }
        p = &(*p)->next;
    } while (q);
    flag? 0 : -1;
}
```

Implementing monster *mhash_find(Mhash *hash, char *name)

```c
monster *mhash_find(Mhash *hash, char *name)
{
    int weight;
    NODE_T *p, *value = NULL;
    BUCKET_T *bucket;

    assert(name);
    assert(hash);
    weight = MOD(mhash_new(name), hash->monsteriness);
    bucket = &hash->buckets[weight];

    p = bucket->list;
    while (p) {
        if (mhash_new_equal(name, &p->name)) {
            value = p;
            break;
        }
        p = p->next;
    }
    return monster;
}
```

The big-O times are the worst case for the add, remove and find functions
Worst-case times of O(n) are evident in the above-mentioned functions this is because of the loops that they have and the add function additionally implements an algorithm which has a worst-case time of O(n)

When is it reflective of the real world?
It is reflective of the real world in that for instance the worst case times are expressed in terms of the big -O in our case O(n) is used to measure the performance of sorting algorithms how much time and space the algorithm takes which can translate to little or much time taken to produce results required by a user of the system.

When is the performance in the real world better?
Performance of an algorithm in the real world may be better in the scenario where for instance a search algorithm is used in highly specialized applications or research purposes is measured in terms of big-o terms which is perfectly fine but the input, the datasets are from a simulation and not the actual real world this may lead to poor performance of the algorithm in the research but works well when real and actual data is used for its input.