1. (3 marks) T. Standish says the following about data structures, such as lists, files, arrays and strings:
They are fictional entities that we create in our imaginations. They provide no essential capacity that cannot be implemented in terms of machine primitives by using assembly language directly.
Then why do computer scientists use data structures? **Briefly, give 3 very different reasons:**

1 - holding more than one property.

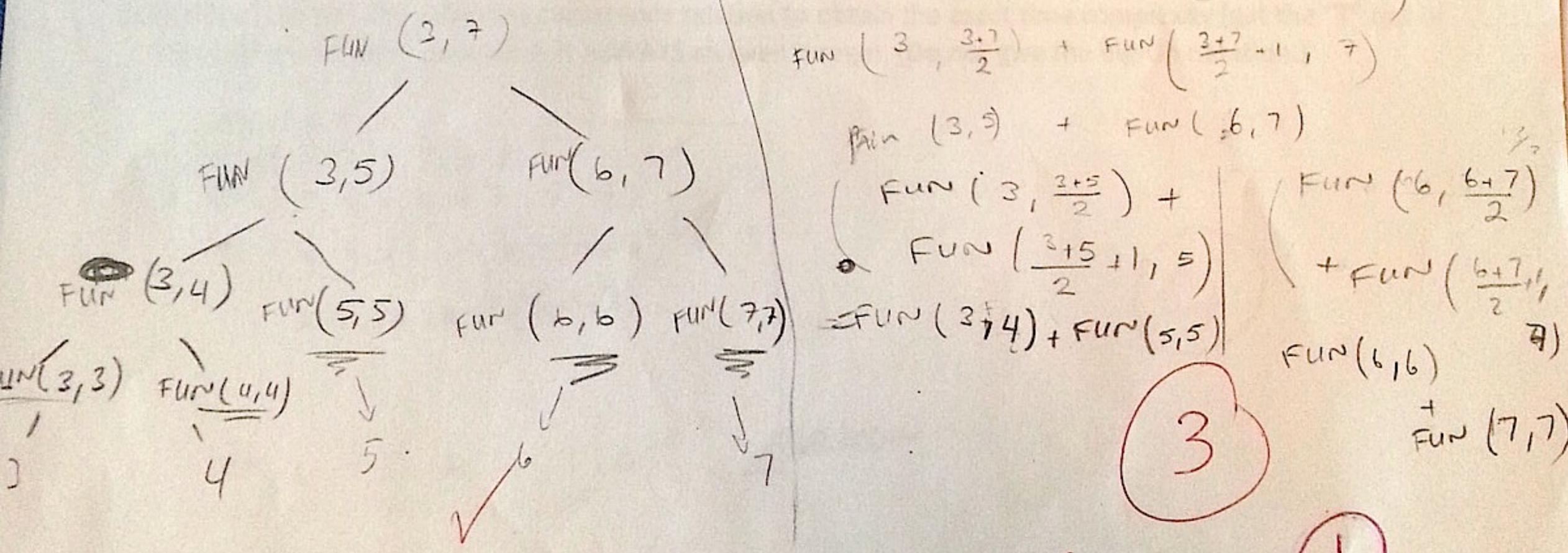
2 - links elements.

3 -

2

2. (3 marks) **Give the call tree for `FUN(3,7)`, where the `FUN` function is as follows:**

```
int FUN ( int x, int y ) {
     if ( x==y )  return (y) ;
     else return (  FUN ( x, (x+y)/2) + FUN ( (x+y)/2+1, y  ) );
}
```



3

3. (2 marks) (a) Is `FUN` above tail-recursive? ☐ Yes ☒ No ✓

1

(b) Briefly, explain your answer:

No, Tail-recursive is when first operation is done last. In this function, the result of a function is not been used by the previous function call.

4. (5 marks) Suppose a main function calls `Rev ( S, 0, x )` to reverse string `S`, where `x` is the last index of `S` before '\0'. i.e., to reverse S="ABCDE", main calls `Rev (S,0,4)`. Assume main always calls `Rev` with an `S` that is non-empty, with `m=0`, and with correct `n` to reverse `S` completely. Finish function `Rev`, using an **Edges and Center recursive solution**.

```
void Rev ( char *S, int m, int n ) {
    char *temp = NULL;
    if ( m==n )
    {       exit(); return; }
    temp = S[m];
    S[m] = S[n];
    S[n] = temp;
    Rev ( S, (m+1), (n-1) );
}
```

what about m>n?

4

temp = 'A'
S[m] = 'E'
S[n]

S = A B C D E
0 1 2 3 4
m n

5. (4 marks) **Unroll** the following recurrence relation to obtain the exact time complexity (get the "T" out of the right-hand side). Assume `n` is ALWAYS an even integer. (**Do not** give the big-Oh notation.)

$T(0) = k$

$T(n) = k + T(n-2)$

$T(n) = k + k + T(n-4)$

$= k + k + k + T(n-6)$

$= k + k + k + k + T(n-8)$

4

$T(n) = \frac{n}{2}k + T(0)$ ← k ← we know

$= \frac{n}{2}k + k$

$= k\left(1 + \frac{n}{2}\right)$ X

$= \frac{3n}{2}k$ → linear.

6. (4 marks) Find the **time** complexity of the following `Mystery` function, where A is a square matrix of dimension `dim` x `dim`. **(Do not** give the big-Oh notation.) Show your work.

```
void Mystery ( int **A, int dim) {
    int i, j ;
    for ( i=1; i<dim; i++ )
        for ( j=0; j<i; j++ )
            A[i][j] = A[j][i];
}
```

$T(n) = An(Bn(C))$ → n here is really the 'dim'

$= An(Bnk)$

$= An \times nk'$

$= \underbrace{Ak'}_{k''}(n^2)$

$= k''n^2$

(2)

7. (2 marks) Find the **space** complexity for the `Mystery` function above. Then give the big-Oh notation. Show your work.

A → for the function,
B → for loop 1.
C → for loop 2. → called Bn times.
(D & E) → for each array → called Cn times. that called Bn times

$S(n) = A + B + Bn(C + Cn(D+E))$

$= A + B + Bk'n^2$

$= O(n^2)$

(0)

8. (3 marks) C uses a run-time stack to process nested function calls. For each call, a stack frame (or activation record) is pushed onto the stack. List **4 items** that are contained in a stack frame.

- operators
- primitive values
- parentheses
-

2.5

9. (4 marks) Suppose a Queue is implemented with linked allocation, using the following definitions and functions:

```
typedef struct QueueNodeTag {
        ItemType                    Item;
        struct  QueueNodeTag  *Link;
} QueueNode;

typedef  struct {
        QueueNode  *Front;
        QueueNode  *Rear;
}Queue;

void InitializeQueue(Queue *Q){
        Q->Front = NULL;
        Q->Rear  = NULL;
}
int Empty(Queue *Q){
        return (Q->Front == NULL && Q->Rear == NULL);
}
```

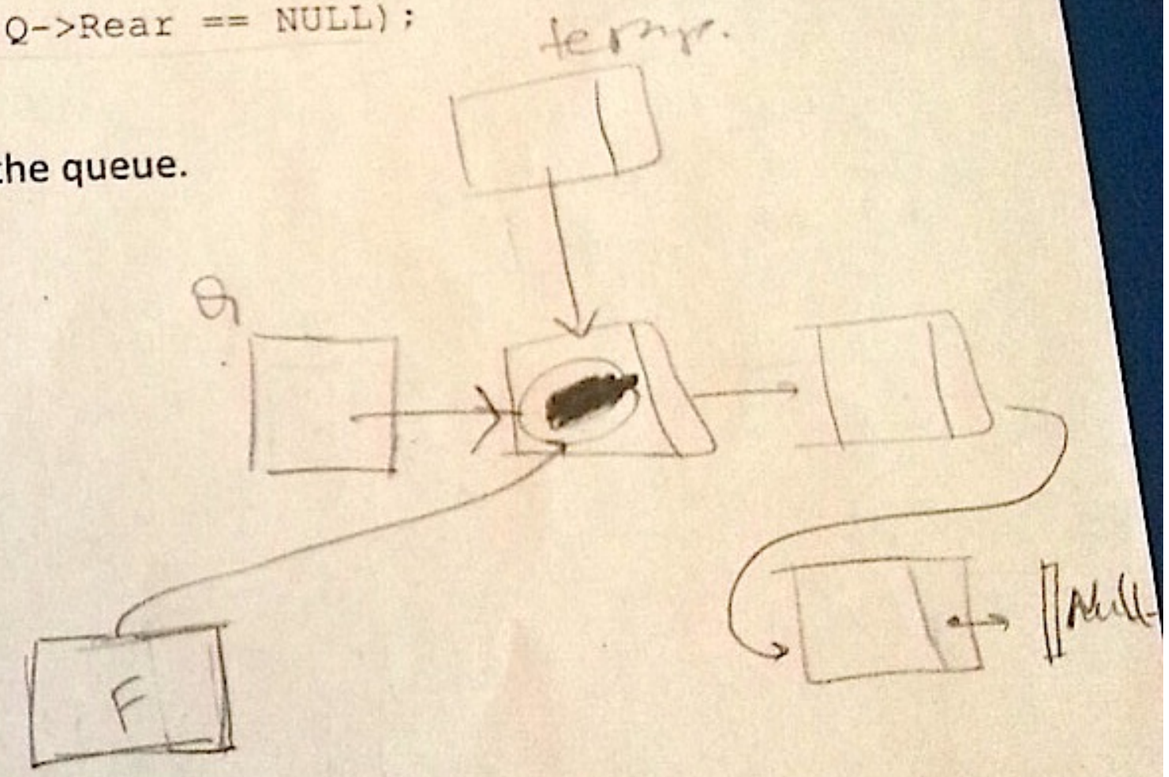
Complete the function below to remove an item from the queue.

```
int Remove ( Queue *Q, ItemType *F ){
   QueueNode *Temp;
   if ( Empty(Q) ) {
        Underflow();
        return 0;
   }
   *F = Q->Front->Item;
```

Temp = F;

Q->Front = Q->Front->Link;

free (Temp);



2