Replace these recurrence relations with simpler ones that would give the same time complexity asymptotically.

For all of these assume $T(1)=1$.

(a) $T(n) = 3n^2 + 4n - 7 + T(n/2)$

(b) $T(n) = 2n + 10\log_2(n) + 100 + 2T(n/2)$

(c) $T(n) = n - 5 + T(1) + T(n-1)$

(d) $T(n) = 23 + T(n-1) + 2T(n/2)$

(e) $T(n) = 7 * 2^n + 7 * n^2 + T(n-1)$

1. Prove that

$f(n) = 3n^2 - 8n - 27 \in \Theta(n^2)$.

2. Set up a recurrence V(n) that gives the value of mystery(n):

```
int mystery(int n)
{
    if (n==1) return(4);
    else return( 5*n + 3 + mystery(n-1));
}
```

Revised deadlines:
Assignment 2A: Tuesday Oct. 8, beginning of class on paper.

Uploads to connex (coming soon):
Assignment 1B Resubmission: Fri. Oct. 11.
Assignment 2B: Tues. Oct. 15.

Assignment 1A will be graded out of 90 (you can have > 100%).

Prove n^3 is not in $O(n^2)$.

Hint: Try a proof by contradiction.

Proof

Assume that n^3 is in $O(n^2)$.

Then by the definition of Big Oh, this means that…

Space Requirements

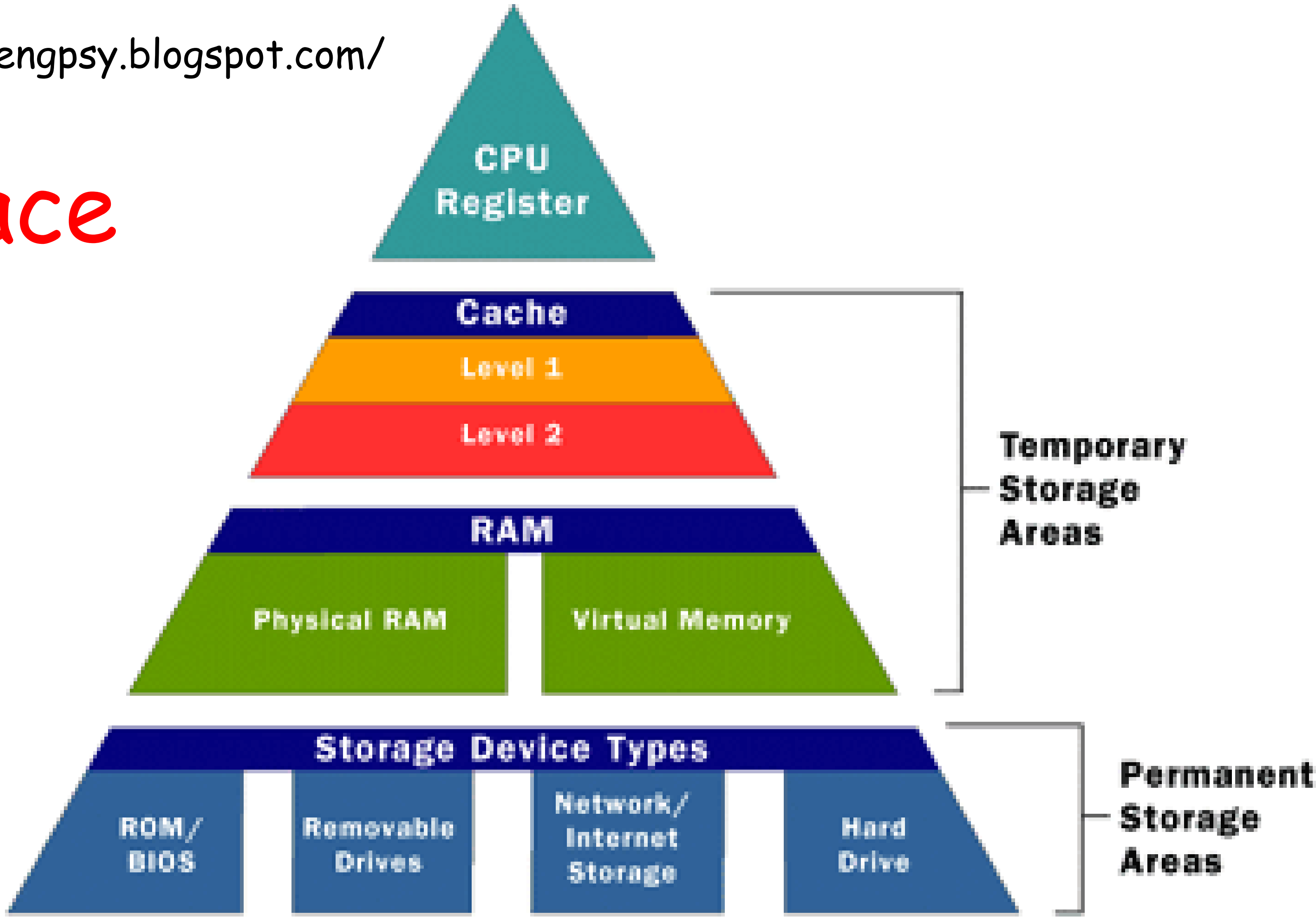
For very big problems, it is sometimes necessary to choose algorithms which do not require too much extra space to avoid crashing the computer.

Algorithms using less space can often run faster than those requiring more space. This is very important for interactive systems such as video games.

We will learn to analyze extra space requirements using the same notation we are using for the running times (Big Oh, Ω and Θ).

http://engpsy.blogspot.com/

Space



L1 cache - 10 nanoseconds, 4-16 kilobytes

L2 cache - 20-30 nanoseconds, 128-512K

Main memory - 60 nanoseconds, 32 MB or more

Extra Space:

For sorting algorithms:

We will only count the space used for the bookkeeping that the algorithm does and not the space that the data is stored in.

Array A with the data: not counted.

Linked list: extra space is used by next pointers.

In the worst case, how much extra space might the program need at one moment in time?

Iterative MaxSort:

```
public class Array
{  int n;  int [] A;
// Space for data in A is not counted
   public void maxSort()
   {   int i, t, end, max_pos;

       for (end= n-1; end > 0; end--)
       {   max_pos=0;
           for (i= 1; i <= end; i++)
               if (A[i] >= A[max_pos])  max_pos= i;
           t= A[max_pos];
           A[max_pos]= A[end];
           A[end]= t;
       }
   }
```

Implicit variable: this

Recursive MaxSort:

Implicit variable: this

```
public void maxSort(int size)
{     int i, t, maxPos;

      if (size <= 1) return;
      maxPos=0;
      for (i=1; i < size; i++)
            if (A[i] >= A[maxPos]) maxPos=i;
      t= A[maxPos];
      A[maxPos]= A[size-1];
      A[size-1] = t;
      maxSort(size-1);
}
```

The extra space used is:

Iterative MaxSort: $\theta(1)$

Recursive MaxSort: $\theta(n)$

If space is an issue, iterative MaxSort is a better choice.

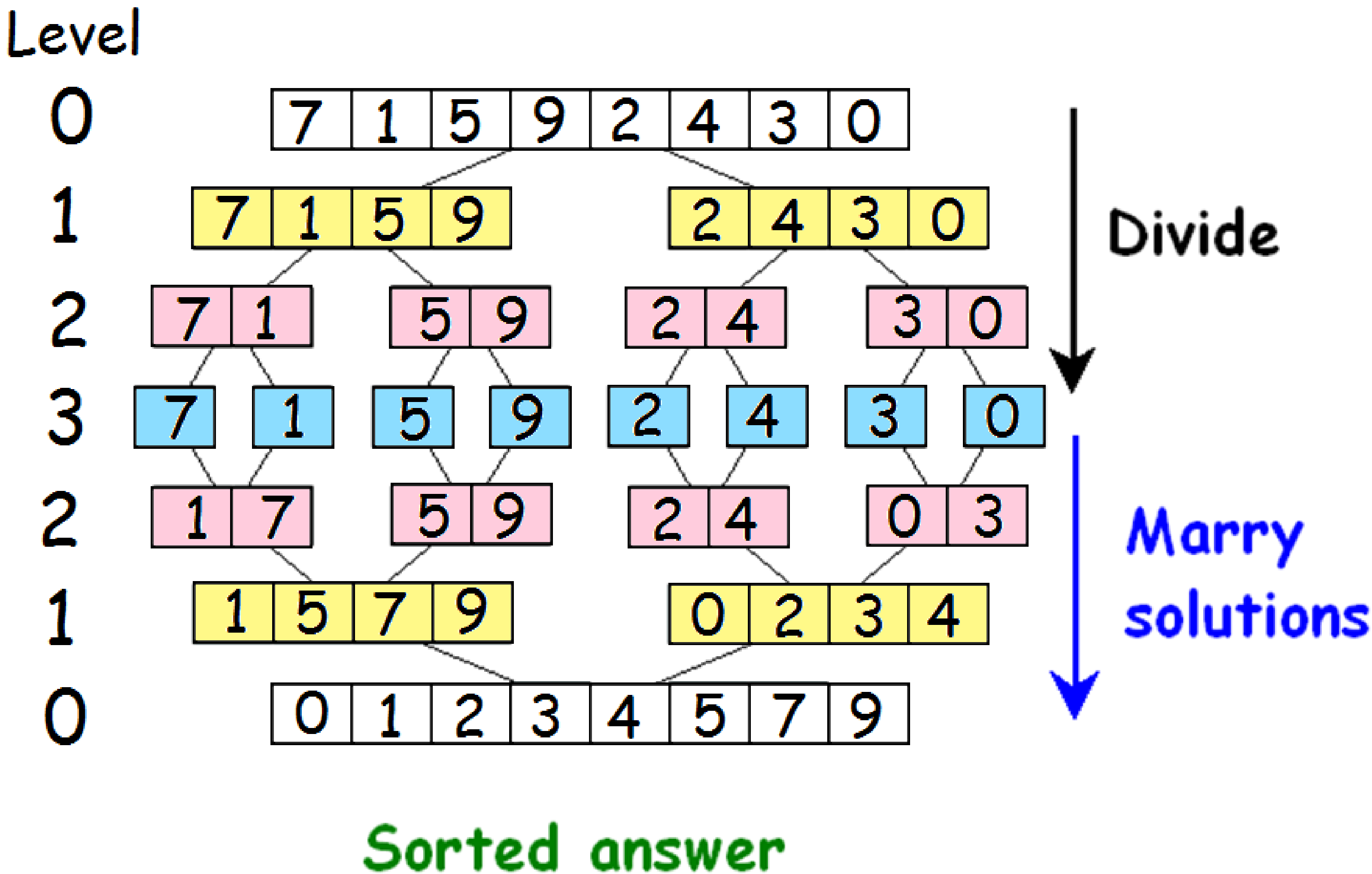
How much extra space does Quicksort use (the implementation presented in class)?

```
// quicksort A[left] to A[right]

public static void quicksort(
                    int[] A, int left, int right)
{
      if (right <= left) return;

      int pivot_pos = partition(A, left, right);

      quicksort(A, left, pivot_pos-1);

      quicksort(A, pivot_pos+1, right);
   }
```

```
// partition A[left] to A[right]
private static int partition(
                int [] A, int left, int right )
{
int i = left; int j = right-1;

while (true)
{
    while (A[i]  < A[right]) {i++;}
    while (A[right] < A[j]) {j--; if (j == left) break; }
    if (i >= j) break;
    swap(A, i, j); i++; j--;
}
swap(A, i, right); // Put pivot element into place
return i;
}
```

How much extra space does Mergesort use if the data structure is a linked list?

Mergesort


Level
0
1
2
3
2
1
0
7 1 5 9 2 4 3 0
7 1 5 9
2 4 3 0
7 1
5 9
2 4
3 0
7
1
5
9
2
4
3
0
1 7
5 9
2 4
0 3
1 5 7 9
0 2 3 4
0 1 2 3 4 5 7 9
Divide
Marry
solutions
Sorted answer

```
public class LinkedList // Same as Lab 1
{  int n;
   ListNode start;
   ListNode rear;

   public void mergeSort(int level)
   {
      LinkedList list1;
      LinkedList list2;
      int i;

/*    A list of size 0 or 1 is already sorted. */

      if (n <=1) return;
```

Implicit variable this.

Code omitted here.

I am assuming for this program, the list is split into two sublists list1 and list2 the same way you do it for your reverse method. The list1 has the first floor(n/2) items and the list2 has the next ceiling(n/2).

Make sure that in your code:

list1.start, list1.rear, list1.n, and

list2.start, list2.rear, list2.n all have correct values and that both list1 and list2 are null terminated.

```
/*     Sort the 2 sublists recursively: */

     list1.mergeSort(level+1);

     list2.mergeSort(level+1);
```

```
/*    Merge the two sorted sublists. */
start= null; rear= null;
LinkedList tmp; // Keeps track of list with smallest key
while (list1.start != null && list2.start != null)
{
      if (list1.start.data < list2.start.data)
            tmp= list1;
      else tmp= list2;

      if (start == null) start= tmp.start;
      else rear.next= tmp.start;

       rear= tmp.start;

     tmp.start= tmp.start.next;
     rear.next= null;
}
```

```
//    Now append the list that still has
//    items in it to the end.

      if (list1.start != null)
            tmp= list1;
       else
            tmp = list2;
       rear.next= tmp.start;
// Make sure our
// object has a correct rear pointer
       rear= tmp.rear;

} // end mergeSort
```