CS2700 : Programming and Data Structures.

WEEK 5

Goal for Today :

- Analyse binary search
- Euclid's Algo
- Write recurrence relations (for running time)

BACK TO BASICS : BINARY SEARCH

```
int binsearch (int A[], int key)
    low = 0; high = n-1;
    while (low ≤ high)
        mid = (low + high)/2
        if A[mid] == key
            return mid;
        if A[mid] < key
            low = mid+1;
        else high = mid -1;
}
```

ANALYSIS:

EACH ITERATION TAKES $O(1)$ TIME

BOUND ON # of ITERATIONS:

Before entering loop
high - low = n-1

When function returns
high - low ≥ -1

Each iteration high - low reduces to at least half its value in prev iteration

$O(\log(n))$ iterations are needed in worst case

BACK TO BASICS : WRITING RECURRENCE RELATIONS.

BIN SEARCH

Check mid

If not found

check one of

left OR right parts

$T(n) = T(n/2) + O(1)$

LIN SEARCH 1

Check mid

If not found

check both left

and right parts

$T(n) = 2T(n/2) + O(1)$

LIN SEARCH 2

Check first element

If not found

check array of size

$n-1$

$T(n) = T(n-1) + O(1)$

FINDING MAX IN AN UNSORTED ARRAY

(1)

m_1 is max in array of size $n-1$

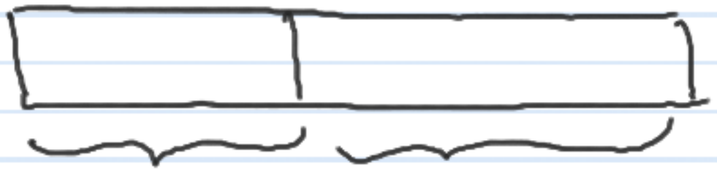
return max (A[low], m_1)

$T(n) = T(n-1) + O(1)$

Depth of recursion is $O(n)$

---

(2)



m_1 → max in array from low, .. mid

m_2 → max in array from mid+1 , .. high.

$T(n) = 2T(n/2) + O(1)$

Depth of recursion is $O(\log n)$.

BACK TO BASICS : EUCLID'S ALGO.

```
int gcd (x, y)
  // assume x >= y
  while (x mod y != 0) {
        x = x mod y;
        swap (x,y)
  }
  return y;
```

$O(k^c)$

size of input $\leq 2k$

$\log x = k$

constant

$x_0 \quad y_0$

$x_1 \quad y_1 = x_0 \bmod y_0$

$x_2 \quad y_2$

$O(k)$ iterations

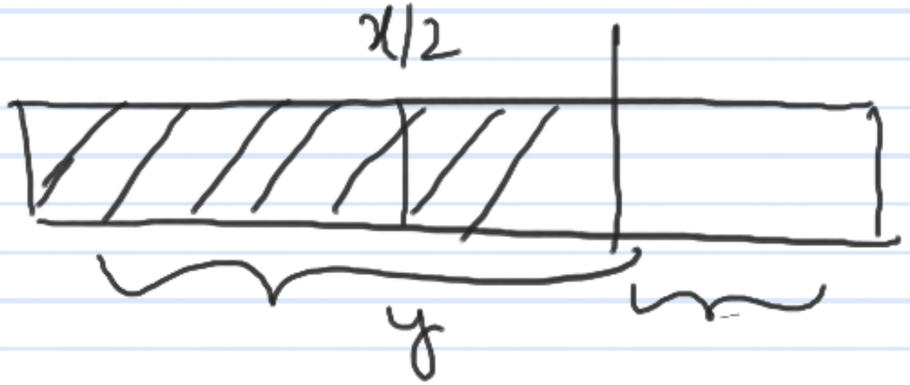
$O(k^c)$ $\qquad$ $O(k^{c+1})$

EUCLID'S ALGO : ANALYSIS.

CLAIM: If $x \geq y$ then $x \bmod y \leq x/2$

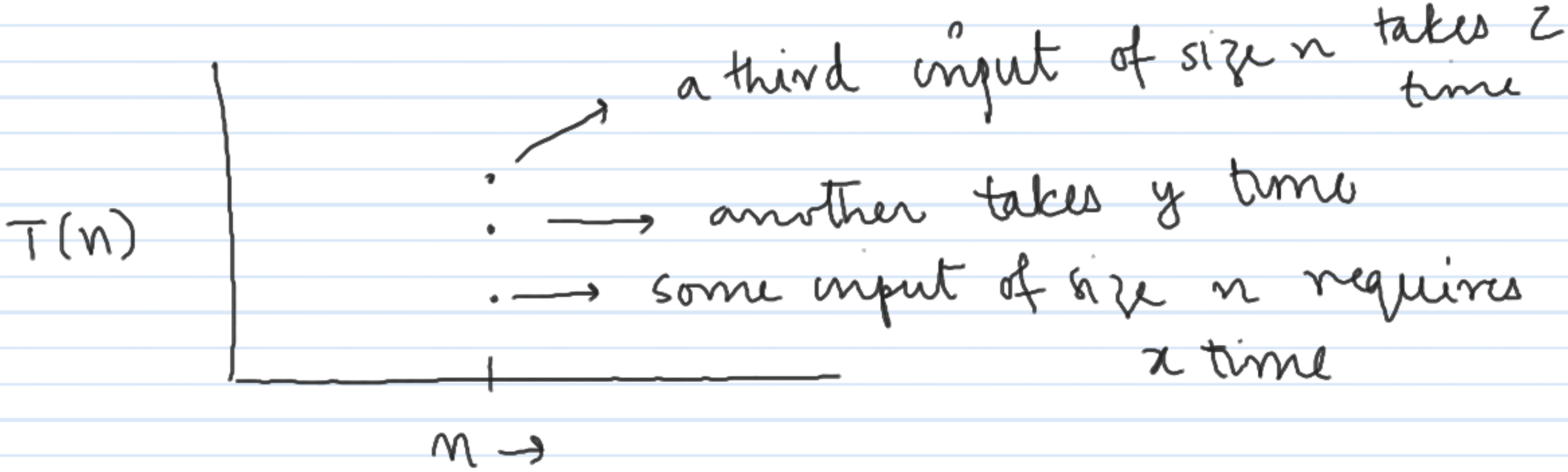
CASE 1 : $y \leq x/2$ | $x \bmod y < y \leq x/2$

CASE 2 : $y > x/2$



WHAT HAPPENS AFTER 2 ITERATIONS

BEST CASE versus WORST CASE



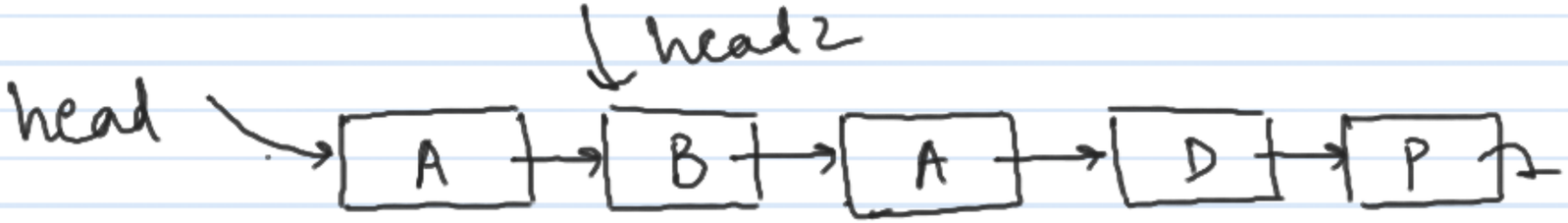
worst case time corresponds to the longest time taken by the program/function on input of size n.

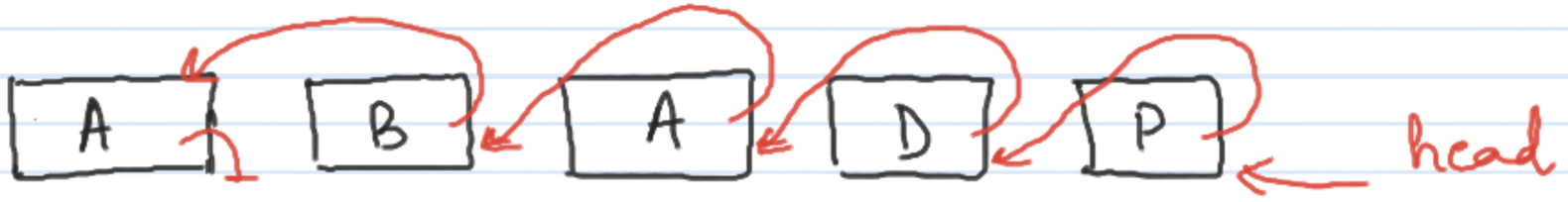
KEY TAKEAWAY:

AN ALGO RUNS IN $O(\log n)$ ~~time~~ iterations if in constant number of iterations it reduces the size of input by a constant factor. (typically 1/2)

AN ALGO RUNS IN $O(N)$ ~~time~~ iterative if in constant time it cuts down the size by a constant such as make n to $n-1$.

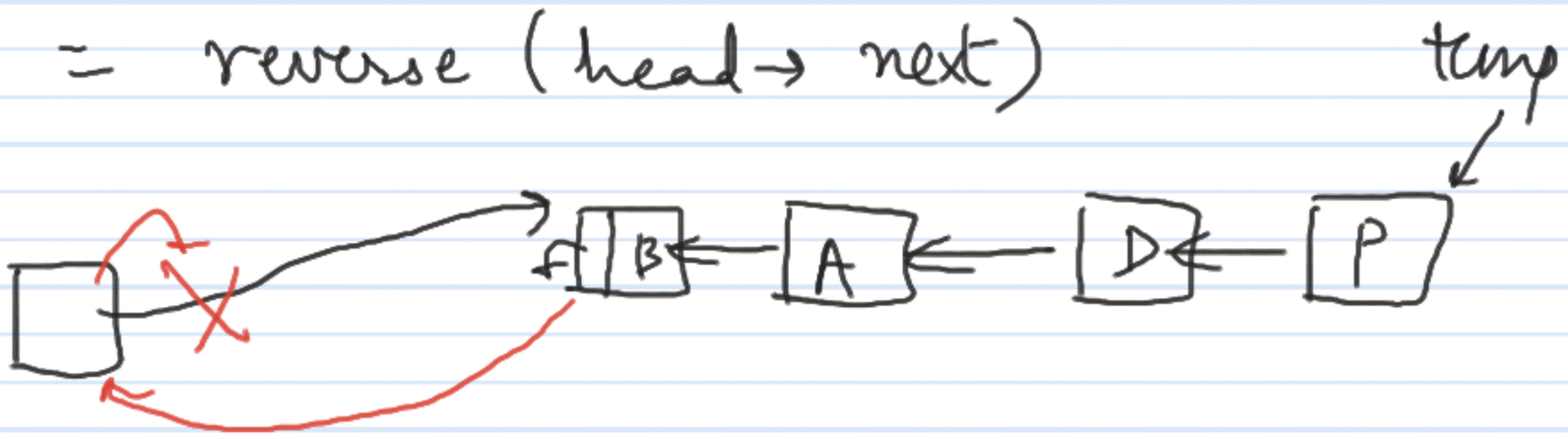
BACK TO LINKED LISTS : REVERSAL



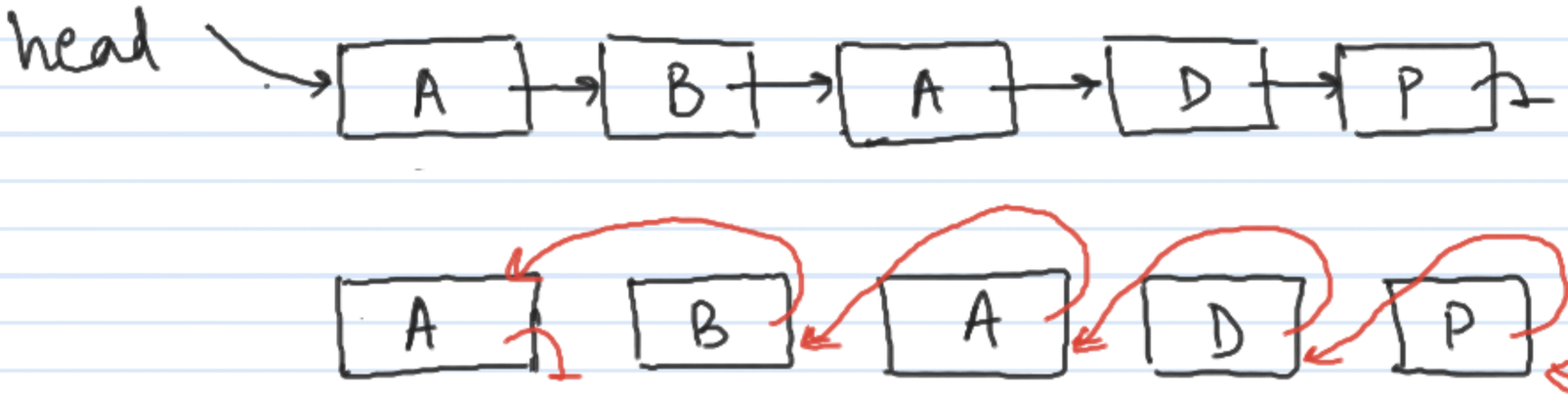


Base case : (head == NULL) or (head → next == NULL)

temp = reverse (head → next)



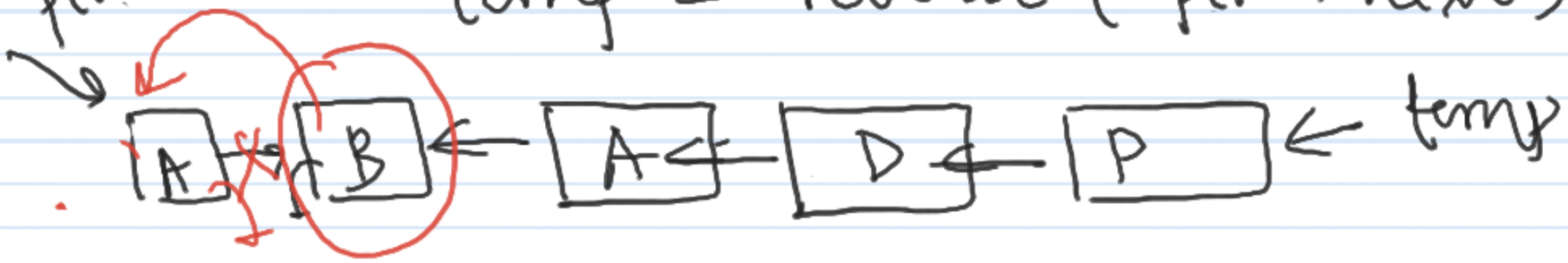
BACK TO LINKED LISTS : REVERSAL



$T(n) =$

$T(n-1)+O(1)$

```
Node* reverse ( Node * ptr )
    if (ptr==NULL || ptr->next==NULL) return ptr
    temp = reverse ( ptr->next);
```

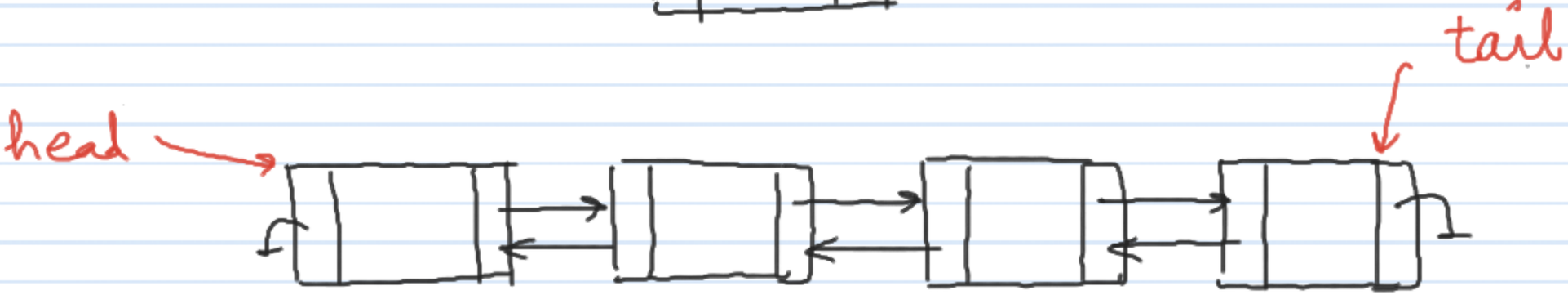


(ptr->next)->next = ptr

ptr->next = NULL

return temp;

DOUBLY LINKED LIST



Operations :

insert : to front as well as at end take O(1)
delete : O(n) } implement these
reverse : O(n). }

DOUBLY LINKED LIST : INSERT