Xuleta Examens

January 31, 2019

```
In [1]: def fib1(n):
            if n==1:
                return n
            if n==0:
                return n
            return fib1(n-1) + fib1(n-2)
```

```
In [4]: def fibonacci(n):
            a, b = 0, 1
            for i in range (1, n+1):
                a,b = b, a+b
            return a
```

```
In [6]: def mul(x,y):
            import math
            if y == 0:
                return 0
            z = mul (x, math.floor(y/2))
            if y%2 == 0:
                return 2*z
            else:
                return x+2*z
```

```
In [8]: def binconvert(n):
            barray = [ ]
            if n == 0:
                return 0
            while n:
                barray.append(n%2)
                n //=2
            barray.reverse()
            return barray

        def modexp(a,n,m):
            bits = binconvert(n)
            solution = 1
            for x in bits:
                solution = (solution*solution)%m
```

```
            if x:
                solution = (solution*a)%m
        return solution
```

In [14]:
```
def lev_distance(first, second):
    if len(first) > len(second):
        first, second = second, first
    if len(second) == 0:
        return len(first)
    first_length = len(first) + 1
    second_length = len(second) + 1
    distance_matrix =[[0] * second_length for x in range(first_length)]
    for i in range(first_length):
        distance_matrix[i][0] = i
    for j in range(second_length):
        distance_matrix[0][j]=j
    for i in range(1, first_length):
        for j in range(1, second_length):
            deletion = distance_matrix[i-1][j] +1
            insertion = distance_matrix[i][j-1] +1
            substitution = distance_matrix[i-1][j-1]
            if first[i-1] != second[j-1]:
                substitution += 1
                distance_matrix[i][j] = min(insertion,deletion, substitution)
    return distance_matrix[first_length-1][second_length-1]
```

In [26]:
```
def quick_sort(A):
    quick_sort_r(A,0,len(A)-1)

def quick_sort_r(A , first, last):
    if last > first:
        pivot = partition(A, first, last)
        quick_sort_r(A, first, pivot - 1)
        quick_sort_r(A, pivot + 1, last)

def partition(A, first, last):
    sred = (first + last)//2
    if A[first] > A [sred]:
        A[first], A[sred] = A[sred], A[first]
    if A[first] > A [last]:
        A[first], A[last] = A[last], A[first]
    if A[sred] > A[last]:
        A[sred], A[last] = A[last], A[sred]
    A [sred], A [first] = A[first], A[sred]
    pivot = first
    i = first + 1
    j = last
    while True:
```

```
            while i <= last and A[i] <= A[pivot]:
                i += 1
            while j >= first and A[j] > A[pivot]:
                j -= 1
            if i >= j:
                break
            else:
                A[i], A[j] = A[j], A[i]
        A[j], A[pivot] = A[pivot], A[j]
        return j
```

In [34]:
```
def mergesort(list):
    if len(list) < 2:
        return list
    else:
        middle = len(list) // 2
        left = mergesort(list[:middle])
        right = mergesort(list[middle:])
    return merge(left, right)

def merge(left, right):
    result = [ ]
    i ,j = 0, 0
    while(i < len(left) and j < len(right)):
        if (left[i] <= right[j]):
            result.append(left[i])
            i = i + 1
        else:
            result.append(right[j])
            j = j + 1
    result += left[i:]
    result += right[j:]
    return result
```

In [41]:
```
def recbinsearch(x, nums, low, high):
    if low>high:
        return -1
    mid = (low + high) // 2;
    items = nums[mid]
    if items == x:
        return mid
    elif x < items:
        return recbinsearch(x,nums,low,mid-1)
    else:
        return recbinsearch(x,nums,mid+1,high)
```