DataBase

June 18, 2024

[9]:

```
[]
```

[ ]:

```
def bubbleSort(arr:list[int]):
    """
    Sorts a list of integers in ascending order using the bubble sort algorithm.
    Args:
        arr (list[int]): The list of integers to be sorted.
    Returns:
        None: The input list is sorted in place.
    """
    for i in range(len(arr)-1):
        sorted=True
        for j in range(len(arr)-1-i):
            if arr[j]>arr[j+1]:
                arr[j],arr[j+1]=arr[j+1],arr[j]
                sorted=False
        if sorted:
           arr=[]

a=[1,2,35 ,4,15,6,7]
print(a)
bubbleSort(a)
print(a)
```

[ ]:

```
def selectionSort(arr:list[int]):
    """
    Sorts a list of integers in ascending order using the selection sort␣
 ↪algorithm.
    Args:
        arr (list[int]): The list of integers to be sorted.
    Returns:
        None: The input list is sorted in place.
    """
    for i in range(len(arr)-1):
        max_index=0
        for j in range(1,len(arr)-i):
```

```
            if arr[j]>arr[max_index]:
                max_index=j
        arr[len(arr)-i-1],arr[max_index]= \
        arr[max_index],arr[len(arr)-i-1]

ar = [87,23,15,3,99,-3]
selectionSort(ar)
print(ar)
```

[ ]:
```
def insertionSort(lst:list[int]):
    """
    Sorts a list of integers in ascending order using the insertion sort␣
 ↪algorithm.
    Args:
        arr (list[int]): The list of integers to be sorted.
    Returns:
        None: The input list is sorted in place.
    """
    for i in range (1,len(lst)):
        tmp=lst[i]
        j=i
        while j>0 and lst[j-1]>tmp:
            lst[j]=lst[j-1]
            j-=1
        lst[j]=tmp

arr=[2,17,5,31,11,13,23,19,21,3]
insertionSort(arr)
print(arr)
```

[ ]:
```
def linearSearch(lst: list[int], key: int) -> int:
    """
    Performs a linear search for a key in a list of integers.

    Parameters:
    lst (list[int]): The list of integers to search.
    key (int): The key to search for.

    Returns:
    int: The index of the key if found, otherwise -1.
    """
    for index, value in enumerate(lst):
        if value == key:
            return index
    return -1

# Example usage:
```

```
numbers = [4, 2, 9, 7, 5, 6]
key = 7
result = linear_search(numbers, key)

if result != -1:
    print(f"Key found at index: {result}")
else:
    print("Key not found in the list.")
```

[ ]:
```
def binariSearch(lst: list[int], key: int) -> int:
    """
    Performs a binary search on a sorted list of integers to find the index of␣
 ↪the target value.
    Args:
        arr (list[int]): A sorted list of integers to search within.
        target (int): The integer value to search for in the list.

    Returns:
        int: The index of the target value in the list if found, otherwise -1.
    """
    low,height = 0,len(lst) - 1
    while low<=height:
        mid = (low+height)//2
        if key < lst[mid] :
            height = mid-1
        elif key > lst[mid]:
            low=mid+1
        else:
            return mid
    return -1

# Example usage:
arr=[2,17,5,31,11,13,23,19,21,3]
selectionSort(arr)
print(arr)
print("19 is in "+str(binariSearch(arr,19))+" placed")
```

insertionSort

merge

[ ]:
```
def merge(l1:list[int],l2:list[int])->list[int]:
    """
    Merge two sorted lists of integers into one sorted list.

    Args:
```

```
    - l1 (list[int]): A sorted list of integers.
    - l2 (list[int]): Another sorted list of integers.

    Returns:
    - list[int]: A sorted list containing all elements from l1 and l2.
    """

    l3=[]
    i=j=0
    while i<len(l1) or j<len(l2):
        if i<len(l1) and j<len(l2):
            if l1[i]<l2[j]:
                l3.append(l1[i])
                i+=1
            else:
                l3.append(l2[j])
                j+=1
        elif i<len(l1):
            l3.append(l1[i])
            i+=1
        else :
            l3.append(l2[j])
            j+=1
    return l3

# Example usage:
ar1=[2,5,8,11]
ar2=[1,5,9,10,11,100]
print(merge(ar1,ar2))
```

Recursion is a programming technique where a function calls itself in order to solve a problem. The main idea is to break down a problem into smaller, more manageable sub-problems of the same type. A recursive function typically has two main components:

Base Case: The condition under which the function stops calling itself, preventing infinite recursion. Recursive Case: The part of the function where it calls itself with a smaller or simpler version of the original problem.

Recursion can be very powerful for solving problems that can naturally be divided into similar sub-problems, such as mathematical problems, tree traversals, and certain types of sorting and searching algorithms.

[ ]:
```
def factorial(n: int) -> int:
    """
    Returns the factorial of a given number.
```

```
    Args:
        n (int): A non-negative integer whose factorial is to be computed.

    Returns:
        int: The factorial of the given number.
    """
    if n == 0 or n==1:
        return 1
    else:
        return n * factorial(n - 1)

# Example usage
print(factorial(5))  # Output: 120
```

[1]:
```
def fibonacci(n: int) -> int:
    """
    Returns the nth Fibonacci number.

    Args:
        n (int): The position in the Fibonacci sequence.

    Returns:
        int: The nth Fibonacci number.
    """
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fibonacci(n - 1) + fibonacci(n - 2)

# Example usage

print(fibonacci(5))  # Output: 8
```

5

[4]:
```
def sum_list(arr: list[int]) -> int:
    """
    Returns the sum of all elements in the list.

    Args:
        arr (list[int]): The list of integers.

    Returns:
        int: The sum of all elements in the list.
    """
    if not arr:
```

```
        return 0
    else:
        return arr[-1] + sum_list(arr[:-1])

# Example usage
print(sum_list([1, 2, 3, 4, 5]))  # Output: 15
```

15

[ ]:
```
# def power(base: int, exponent: int) -> int:
    """
    Returns the base raised to the power of the exponent.

    Args:
        base (int): The base number.
        exponent (int): The exponent.

    Returns:
        int: The result of base raised to the power of the exponent.
    """
    if exponent == 0:
        return 1
    else:
        return base * power(base, exponent - 1)

# Example usage
print(power(2, 3))  # Output: 8
```

MergeSort

[5]:
```
def Merge(a:list[int],b:list[int],c:list[int]):
    """
    Merges two sorted lists a and b into the list c, in sorted order.

    Parameters:
    a (list): The first sorted list.
    b (list): The second sorted list.
    c (list): The list that will contain the merged sorted elements of a and b.

    Returns:
    None
    """
    c.clear()
    while len(a)>0 and len(b)>0:
        if a[0]<= b[0]:
            c.append(a.pop(0))
        else:
            c.append(b.pop(0))
```

```
    if len(a)>0:
        c+=a
    else:
        c+=b
def MergeSort(a:list[int]):
    """
    Sorts a list a using the Merge Sort algorithm.

    Parameters:
    a (list): The list to be sorted.

    Returns:
    None
    """
    if len(a) <=1:
        return
    b=a[:len(a)//2]
    c=a[len(a)//2:]
    MergeSort(b)
    MergeSort(c)
    Merge(b,c,a)

ar=[5,7,11,9,3,5,1,-4,19]
MergeSort(ar)
print(ar)
```

```
[-4, 1, 3, 5, 5, 7, 9, 11, 19]
```

Recursion

[8]:
```
def reverse_string(s: str) -> str:
    """
    Returns the reversed string.

    Args:
        s (str): The string to be reversed.

    Returns:
        str: The reversed string.
    """
    if len(s) == 0:
        return s
    else:
        return s[-1] + reverse_string(s[:-1])

# Example usage
print(reverse_string("hello"))  # Output: "olleh"
```

```
olleh
```

[7]:
```python
def is_palindrome(s: str) -> bool:
    """
    Checks if the given string is a palindrome.

    Args:
        s (str): The string to be checked.

    Returns:
        bool: True if the string is a palindrome, False otherwise.
    """
    if len(s) <= 1:
        return True
    else:
        return s[0] == s[-1] and is_palindrome(s[1:-1])

# Example usage
print(is_palindrome("radar"))  # Output: True
```

True

[6]:
```python
def gcd(a: int, b: int) -> int:
    """
    Returns the greatest common divisor of a and b.

    Args:
        a (int): The first integer.
        b (int): The second integer.

    Returns:
        int: The greatest common divisor of a and b.
    """
    if b == 0:
        return a
    else:
        return gcd(b, a % b)

# Example usage
print(gcd(48, 18))  # Output: 6
```

6

[ ]:
```python
def func(ls:list[int],s:int,e:int)->int:
    if s==e :
        return ls[s]
    if s==e+1:
        return 0

    return ls[s]*ls[e]+func(ls,s+1,e-1)
```

```
# Example usage
l1=[2,3,4,6,8]
print(func(l1,0,len(l1)-1))
```

[ ]: