Tanya Gyanmote
cruiz id - tgyanmot

DESIGN DOC

Main

- Create a function for -H case that prints out the usage of the program
- Create a function that will populate my array with using the seed function, mtrand_seed
    - Have a for loop that will go through the array to populate each element
        - Return the array
- Have the default variables
- Create a set using set_empty
- Create cases for each type of input
    - Case a to print out all the sorts
    - Case s for shell sort
    - Case b for bubble
    - Case q for quick sort
    - Case h for heap sort
    - Case r for the seed
    - Case n for array size
    - Case p for the elements of the array
    - Case h to print out the synopsis
    - In cases, a-h using the set I created use set_insert and have a number according to the sort
- For Bubble sort
    - Reset my stats
    - Create an array while calling my function that populates my array
    - Call bubble sort with the array I just populated
    - Have a temp variable to keep count of when I should create a new line
    - Have a print statement with Bubble sort, elements, moves, compares
    - Have a for loop so the elements of the array could be printed
        - Use the print statement given in the pdf
        - Add 1 to my temp counting when I should add a new line
            - If my temp variable % 5 then its time to add a new line
- For Shell sort
    - Reset my stats
    - Create an array while calling my function that populates my array
    - Call shell sort with the array I just populated
    - Have a temp variable to keep count of when I should create a new line
    - Have a print statement with shell sort, elements, moves, and compares
    - Have a for loop so the elements of the array could be printed
        - Use the print statement given in the pdf
        - Add 1 to my temp counting when I should add a new line
            - If my temp variable % 5 then its time to add a new line
- For Quick sort

    - Reset my stats
    - Create an array while calling my function that populates my array
    - Call quick sort with the array I just populated
    - Have a temp variable to keep count of when I should create a new line
    - Have a print statement with quick sort, elements, moves, and compares
    - Have a for loop so the elements of the array could be printed
        - Use the print statement given in the pdf
        - Add 1 to my temp counting when I should add a new line
            - If my temp variable % 5 then its time to add a new line
- For Heap sort
    - Reset my stats
    - Create an array while calling my function that populates my array
    - Call heap sort with the array I just populated
    - Have a temp variable to keep count of when I should create a new line
    - Have a print statement with heap sort, elements, moves, and compares
    - Have a for loop so the elements of the array could be printed
        - Use the print statement given in the pdf
        - Add 1 to my temp counting when I should add a new line
            - If my temp variable % 5 then its time to add a new line

Shell Sort in Python

```python
def gaps(n):
  while n > 1:
    n = 1 if n <= 2 else 5 * n // 11
    yield(n)

def shell_sort(arr):
    for gap in gaps(len(arr)):
        for i in range(gap, len(arr)):
            j = i
            temp = arr[i]
            while j >= gap and temp < arr[j - gap]:
                arr[j] = arr[j - gap]
                j -= gap
            arr[j] = temp
    return arr
```

Shell

- Using the pseudocode given
- Created a gap junction
    - If gap is less than 2, return the size -1
    - If the gap is less than 0 return 0
    - Else follow the equation from the pseudocode and return size * 5-floor division of 11 to find the new gap
- Shell sort function
    - Create a for loop that goes through the gap function for every step
        - Creating another for loop that loops starting from the gap till the end of the array

- Having a new variable to store i, the gap
- Having a temp variable, where we store the value of the element of the array
- Have a while loop that checks to see if our j, gap is greater than our initial gap from the first for loop and if the temp variable is less than array element of j-gap
    - If it is then change the value
- Change arr[j] to temp

Bubble Sort in Python

```python
def bubble(a):
  for i in range (len(a) - 1):
    swapped = False
    for j in range (len(a) - 1, i, -1):
      if a[j] < a[j-1]:
        a[j], a[j-1] = a[j-1], a[j]
        swapped = True
    if not swapped:
      break
  return a
```

Bubble Sort

- Creating a for loop that going through the array
    - Having a boolean to keep track for true and fals
        - Having another for loop that going through the array but started at the end of the array and decrements
            - If the end of the array is less than the element before it then swap the values
            - Make the boolean true
    - If it's not swapped then break

Quick Sort in Python

```python
from shellsort import shellsort
SMALL = 8

def quicksort(a):
  if len(a) < SMALL:
    return shellsort(a)

  pivot = (a[0] + a[len(a) // 2] + a[-1]) // 3
  left  = [ _ for _ in a if _ <  pivot]
  mid   = [ _ for _ in a if _ == pivot]
  right = [ _ for _ in a if _ >  pivot]
  return quicksort (left) + mid + quicksort (right)
```

Quick Sort

- Using the puesdocode from above
- Have a variable small equal to 8
    - Check if the size of the array is less than 8
        - if it is then call shellsort

- Have a variable for left, right, and pivot
- Set left to 0, start of array
- Set right to n_elements -1, end of array
- Set pivot to the equation that was given above
- Create a while loop to make sure left is always going to be less than right because we want the array least to greatest
- Checking to see if pivot is greater than array[left] or arr[left] is equal to pivot
    - If it is drcrement the left
- Checking to see if pivot is less than array[right]
    - is it is then decrement right by 1
- Check if right is greater than left
    - If it is then swap the left and right elements, arr[left] and arr[right]
- Call quicksort with the first element and left
- Call quicksort with the array of left and size of the array-1

```
def l_child (n):
  return 2 * n + 1

def r_child (n):
  return 2 * n + 2

def parent(n):
  return (n - 1) // 2

def up_heap (a, n):
  while n > 0 and a[n] > a[parent(n)]:
    a[n], a[parent(n)] = a[parent(n)], a[n]
    n = parent(n)

def down_heap (a, heap_size):
  n = 0  # Down heap from root
  while l_child(n) < heap_size:
    if r_child (n) == heap_size:
      bigger = l_child(n) # If there's no right child, the left is bigger
    else:
      bigger = l_child(n) if a[l_child(n)] > a[r_child(n)] else r_child(n)
    if a[n] > a[bigger]:
      break
    a[n], a[bigger] = a[bigger], a[n]
    n = bigger

def build_heap (a):
  heap = [0] * len(a)
  for n in range(len(a)):
    heap[n] = a[n]
    up_heap (heap, n)
  return heap

def heapsort (a):
  heap = build_heap (a)
  sorted_list = [0] * len (a)
  for n in range (len(a)):
    sorted_list[n], heap[0] = heap[0], heap[len(a) - n - 1]
    down_heap (heap, len(a) - n)
  return sorted_list
```

Heapsort

- Creating L child using puesdocode from above
- Creating R child using puesdocode from above
- Creating parent using puesdocode from above
- Create up_heap the same as above but check if a[n] is less than a[parent(n)] because we are trying to find the smallest num, for min heap
- Create downheap using pseudo from above
    - Have n =0
    - Have a while loop to make sure left child is less than heapsize
        - If r child is the heap size
            - Make the smallest value l child

        - Else
            - If l child is less than r child
                - Using the smaller value and setting it to l child
            - Else the smallest value is r child
        - If arr[n] is less than arr[smaller] then the array is sorted so you break
        - Swap the value of arr[n] and arr[smaller] if its not the smallest
        - Change n to smaller
- For buildheap using the puesocode
    - Create a heap array using calloc with n_elements and size of uint32_t
    - Create a for loop that goes through the array
        - Populate the array with arr[n]
        - Call upheap and put in heap,n
    - Return heap
- For heapsort
    - Create a heap array using buildheap
    - Create a for loop that goes through the array
        - Make arr[n] to heap[0]
        - Make heap[0] to heap[n_elements-1]
    - Make sure to free the heap after using