DATA1050 Midterm Exam

22 October 2020

Name: ____Enmin Zhou________________________

Banner ID: _____B01694721__________________

Brown Short Name: ______ezhou24__________

Date: ___________22 Oct 2020_________________

Note: If need be, please box your answers to make clear which part should be graded, otherwise we won't know what to grade.

Brown Short Name:____________________

Python Basics, Searching, TDD

A monotonic sequence is always the same value or increasing or decreasing in the same direction, below you are to write code to detect montonic sequences of numbers stored in an array.

Examples:
[1,2,2,5] is monotonic
[4,2,1,1,-10] is monotonic
[1,5,4] is not monotonic

1. Include docstrings and state the time and space complexity of each of your functions in a comment above the function definition.
2. Share reasoning where it is not clear that the complexity is something like O(1) for looking up an array element, O(1) for a hash-map lookup, O(n log n) for sorting, etc.
3. **No TFD steps other than docstrings and tests, unless you want to include them.**

```
### For L empty and the singleton return false for isMonotonic and isAscending.

def isMonotonic(L): # returns true if list L is monotonic
    # Write your code here.
        """
        I use increase and decrease to check if the function is in a certain monotone
        pattern
        input is a L
        output is true or false
        time: O(N)
        space: O(1)
        special case: L is empty
        normal case: L is not empty
        """
        increase = True
        decrease = True
        for i in range(0, len(L) - 1):
            if L[i] > L[i + 1]:
                increase = False
                break
        for i in range(0, len(L) - 1):
            if L[i] < L[i + 1]:
                decrease = False
                break
```

```
        return increase or decrease

def isAscending(L): # returns true is L is ascending, reuse isMonotonic
    # Write your code here.
        """
        I check if L is Monotonic and the first element is smaller or equal to the last
        element
        space complexity: O(1)
        time complexity: O(N)
        input is a list
        output is true or false
        special case: L is empty
        normal case: L is not empty
        """
        if L is []:
            return True
        return isMonotonic(L) and L[0] <= L[-1]


def test_e1_1():
    # Test cases for new functions above
        assert isMonotonic([3,2,1]) is True
        assert isMonotonic([1,2,3,2,1]) is False
        assert isMonotonic([1,2,3,4,5,6,7,7]) is True
        assert isAscending([1,2,3,44]) is True
        assert isAscending([1,3,4,2]) is False
```

```
##  Below you can assume A and B are already sorted into ascending order.

def ascContains(A, e):
# returns true if A contains e -- use the bisect library
    # Write your code here.
        """
        input is array and an element e
        output is true or false
        special case: e is not in A
        normal case: e is in A
        I use bisect and find if the index is 0, if not 0 compare element on the left of
        index and e
        space complexity: O(1)
        time complexity: O(N)
        """
        index = bisect.bisect(A, e)
        if index == 0:
            return False
        if A[index - 1] == e:
            return True
        else:
            return False


def ascInsert(A, e):
# Inserts element e into A, keep A in ascending order. hint consider using bisect.  Do this
inplace.
    # Write your code here.
        """
        input is array and an element e
        output is modified array
        all case are normal case
        Use bisect.insort to inser e into A
        time complexity: O(n)
        space complexity: O(1)
        """
        bisect.insort(A,e)
        return A
```

```
def ascDelete(A, e):
# Deletes element e from A, hint consider using bisect. Do this inplace.
    # Write your code here.
        input is array and an element e
        output is modified array
        all cases are normal
        I use bisect.bisect to check if any 'e' is still in A and use remove() to remove the
        element
        space complexity: O(1)
        time complexity: O(n)
        """
        while A[bisect.bisect(A, e)] != 0 and A[bisect.bisect(A, e) - 1] == e:
            A.remove(e)
        return A


def test_e1_2():
    # Test cases for new functions above
        assert ascContains([1,2,3], 2) is True
        assert ascContains([3,4,5,6], 2) is False
        assert ascContains([2,2,3,4,5], 2) is True
        assert ascInsert([1,2,3], 2) == [1,2,2,3]
        assert ascDelete([1,2,3,3,3,4], 3) == [1,2,4]
```

```
def ascMerge(A, B):
# hint consider using bisect and a variant of merge. Return the result in ascending order.
    # Write your code here.
        """
        input are two lists
        output i a merged list
        I use the bisect.insort to insert elements in B into A
        all cases are normal
        time complexity: O(n)
        space complexity: O(1)
        """
        for i in B:
            bisect.insort(A, i)
        return A

# Example: ascMerge([1,2],[2,4]) == [1, 2, 2, 4]


def ascUnion(A, B):
# hint consider using bisect and a variant of merge. Return the result in ascending.
    # Write your code here.
        """
        input are two lists
        output i a union list
        all cases are normal
        In a loop through B, I use bisect.bisect to check if element in B is in A and if not
        I insert it.
        time complexity: O(n)
        space complexity: O(1)
        """
        for i in B:
            index = bisect.bisect(A, i)
            if index == 0 or A[index - 1] != i:
                bisect.insort(A, i)
        return A

# Example: ascUnion([1,2],[2,4]) == [1, 2, 4]


def ascIntersection(A, B):
# hint consider using bisect and a variant of merge. Return the result in ascending order.
    # Write your code here.
```

```python
    """
    input are two lists
    output i a union list
    all cases are normal
    In a loop through B, I use bisect.bisect to check if element in B is in A and if not
    I insert it.
    time complexity: O(n)
    space complexity: O(n)
    """
    intersection = []
    for i in B:
        index = bisect.bisect(A, i)
        if index != 0 and A[index - 1] == i:
            intersection.append(i)
    return intersection



def test_e1_3():
    # Test cases for new functions above
    assert ascMerge([1,2,5,6,8,9], [3,4,6]) == [1,2,3,4,5,6,6,8,9]
    assert ascUnion([1, 2, 5, 6, 8, 9], [3, 4, 6]) == [1, 2, 3, 4, 5, 6, 8, 9]
    assert ascIntersection([1, 2, 5, 6, 8, 9], [3, 4, 6]) == [6]
```

Brown Short Name:____________________

e1.2.1 Code Understanding

For the `OrderedSet` class (see next page),

1. Explain what the class does.
2. Provided a detailed explanation of how storage is implemented.
3. Include an example of a three element set.

hints: see the relevant python documentation on collections.abc.Set and collections.abc.MutableSet

Your Answer:

1. The class implements an OrderedSet which supports the operations of set and the elements in the set are ordered in the sequence of elements being added into the set.

2. The storage is implemented using a self.end and a self.map
   self.end: in a list of three elements with the first element being set to None, and the rest two are recursive lists with the first one being ordered in the sequence of adding and the second one being ordered in the reverse of first one.
   self.map: map each added element to its recursive lists of elements before it and after it.
   The total storage is implemented as a doubly linked list in the representation of lists. The size of the storage is modified dynamically if an element is added or deleted from the class (both map and end will be updated).

3. self: OrderedSet(['a', 'b', 'c'])

   self.end: [None, ['c', ['b', ['a', [...], [...]], [...]], [...]], ['a', [...], ['b', [...], ['c', [...], [...]]]]]

   self.map: {'a': ['a', [None, ['c', ['b', [...], [...]], [...]], [...]], ['b', [...], ['c', [...], [None, [...], [...]]]]], 'b': ['b', ['a', [None, ['c', [...], [...]], [...]], [...]], ['c', [...], [None, [...], ['a', [...], [...]]]]], 'c': ['c', ['b', ['a', [None, [...], [...]], [...]], [...]], [None, [...], ['a', [...], ['b', [...], [...]]]]]}

```
from collections.abc import MutableSet

class OrderedSet(MutableSet):

### TODO e1.2.2
### Add an appropriate docstring to each method below
### Add a comment above each method with its and time and space complexity
### set self to iterable, if none, generate an empty list. Both O(1)
  def __init__(self, iterable=None):
      self.end = end = []
      end += [None, end, end] # sentinel node for doubly linked list
      self.map = {}           # key --> [key, prev, next]
      if iterable is not None:
          self |= iterable
### return length of self.map, both O(1)
  def __len__(self):
      return len(self.map)
### return if key is in self.map, space O(1), time O(1)
  def __contains__(self, key):
      return key in self.map
### add a key into self.map, space O(1), time O(1)
  def add(self, key):
      if key not in self.map:
          end = self.end
          curr = end[1]
          curr[2] = end[1] = self.map[key] = [key, curr, end]
```

```
### TODO e1.2.3
### Add an appropriate docstring to each method below
### Add a comment above each method with its and time and space complexity
### discard the key in self.map, both O(1)
  def discard(self, key):
      if key in self.map:
          key, prev, next = self.map.pop(key)
          prev[2] = next
          next[1] = prev
### pop last key in self.map, both O(1)
### raise error if empty
  def pop(self, last=True):
      if not self:
          raise KeyError('set is empty')
      key = self.end[1][0] if last else self.end[2][0]
      self.discard(key)
      return key
```

```python
### TODO e1.2.4
### Add an appropriate docstring to each method below
### Add a comment above each method with its and time and space complexity
### define the '==' operation in OrderedSet, both O(1)
  def __eq__(self, other):
      if isinstance(other, OrderedSet):
          return len(self) == len(other) and list(self) == list(other)
      return set(self) == set(other)
### define the iteration process of OrderedSet, time O(n), space O(1)
  def __iter__(self):
      end = self.end
      curr = end[2]
      while curr is not end:
          yield curr[0]
          curr = curr[2]
```

```
### TODO e1.2.5
### Add an appropriate docstring to each method below
### Add a comment above each method with its and time and space complexity
### define the reverse operation, time O(n), space O(1)
  def __reversed__(self):
      end = self.end
      curr = end[1]
      while curr is not end:
          yield curr[0]
          curr = curr[1]
### define the representation of OrderSet in string python ‘print’. Both O(1)
  def __repr__(self):
      if not self:
          return f'{self.__class__.__name__}()'
      return f'{self.__class__.__name__}({list(self)})'

if __name__ == '__main__':
  s = OrderedSet('shazaam')
  t = OrderedSet('simsalabim')
  print(s)
  print(t)
  print('Union:', s | t)
  print('Intersection:', s & t)
  print('Difference:', s - t)
```

Brown Short Name:____________________

Sorting

Problem e1.3 Implement Counting Sort in Python

```
### Your solution here.  Assume you are given list L and will return the sorted
### sorted version of L.   Assume the values in L are integer values between 0 and
100, inclusive.  Call your function count_sort(L)
"""
Input: a list
Output: a sorted list
Design: we use a count to record the number of integers in the list
And we count the number of each characters in the list
All cases are normal case
Time complexity:O(N)
space complexity:O(N)
"""

def count_sort(L):
    size = len(L)
    output = [0] * size
    max_value = max(L)
    if max_value < size:
        max_value = size
    count = [0] * (max_value + 1)

    for i in range(0, size):
        count[L[i]] += 1

    for i in range(1, max_value+1):
        count[i] += count[i - 1]

    i = size - 1
    while i >= 0:
        output[count[L[i]] - 1] = L[i]
        count[L[i]] -= 1
        i -= 1
    for i in range(0, size):
        L[i] = output[i]
    return L
```

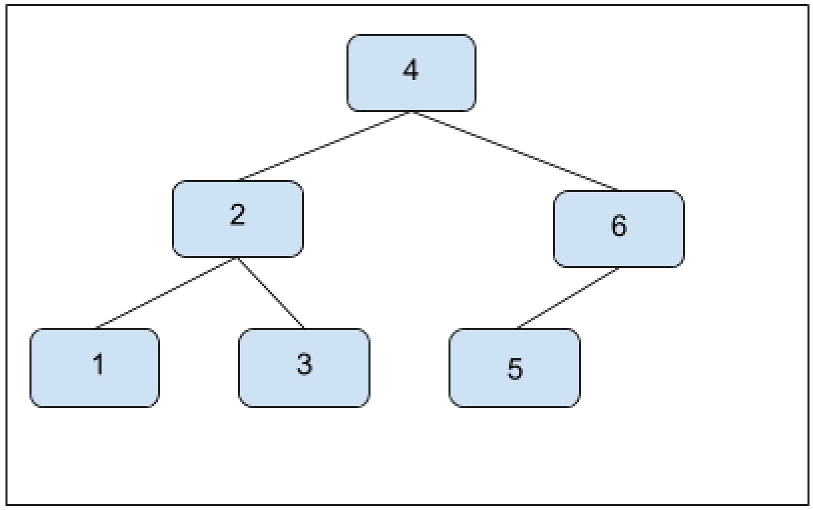
```python
def test_count_sort():
    l = [1,2,1,1,3,5,5,4]
    count_sort(l)
    assert l == [1,1,1,2,3,4,5,5]
    assert count_sort([1,2,4,88,1,46]) == [1,1,2,4,46,88]


if __name__ == '__main__':
    test_count_sort()
```

Tree Definitions, Facts, and Representation

Fun in the forest

Problem e1.4 True or False, explain if false.



__T__ This is a binary tree

__T__ This is a complete tree

__T__ This is a BST

__F__ This is a Heap

__T__ This is an AVL tree

Tree Representations

Prolem e1.5 Represent the tree above using the following

1. list of list notation (or functional notation)

   [4, [2, [1, [], []], [3, [],[]]], [6, [5, [], []], []]]

2. adjacency lists

   [{2, 6}, {1, 3}, {5}, {}, {}, {}]

3. connectivity matrix

[

0 1 0 0 0 0

1 0 1 1 0 0

0 1 0 0 0 0

0 1 0 0 0 1

0 0 0 0 0 1

0 0 0 1 1 0

]

4. array notation

[4, 2, 6, 1, 3, 5]

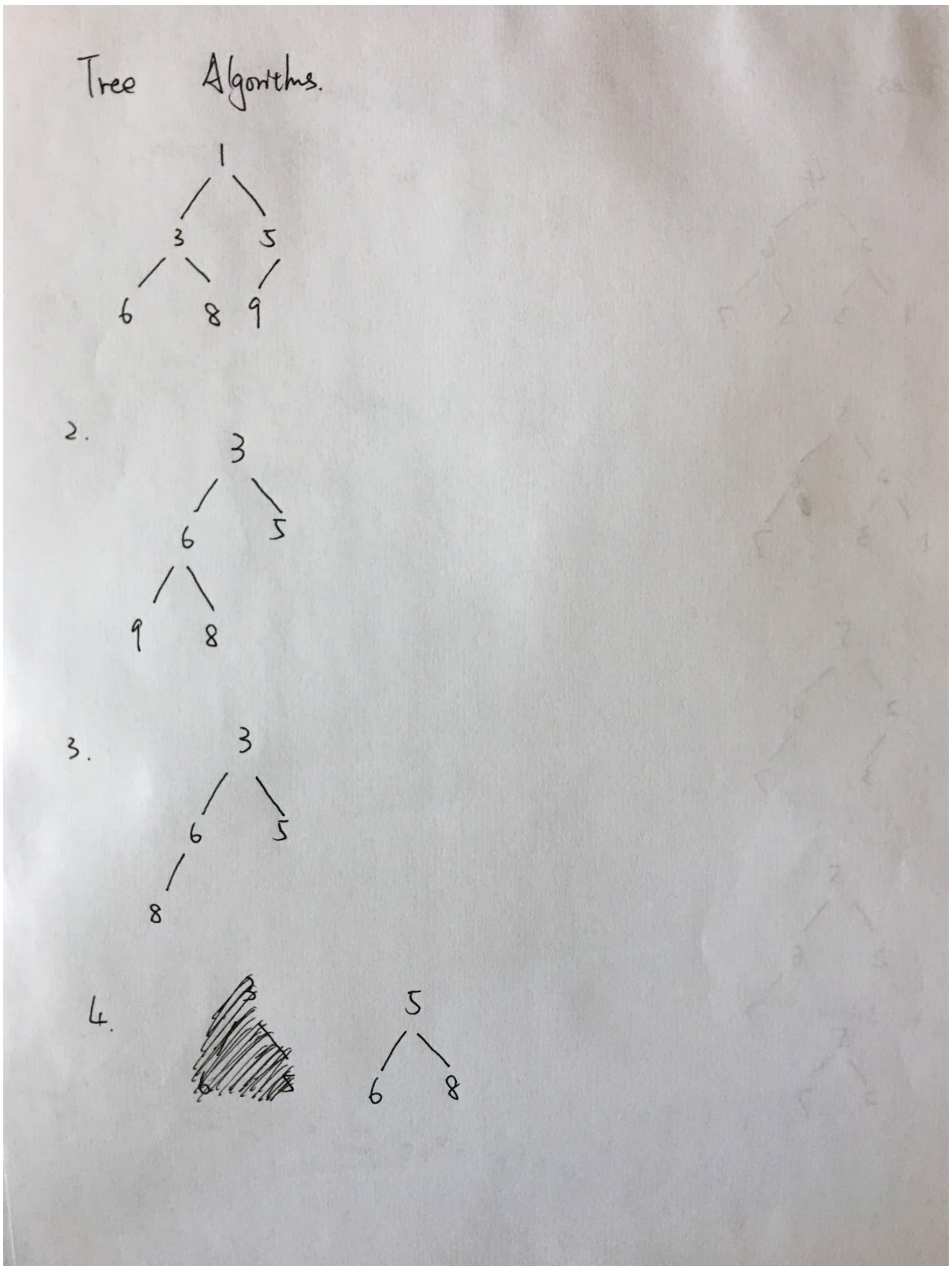
Tree Algorithms

Problem e1.6 Heaps

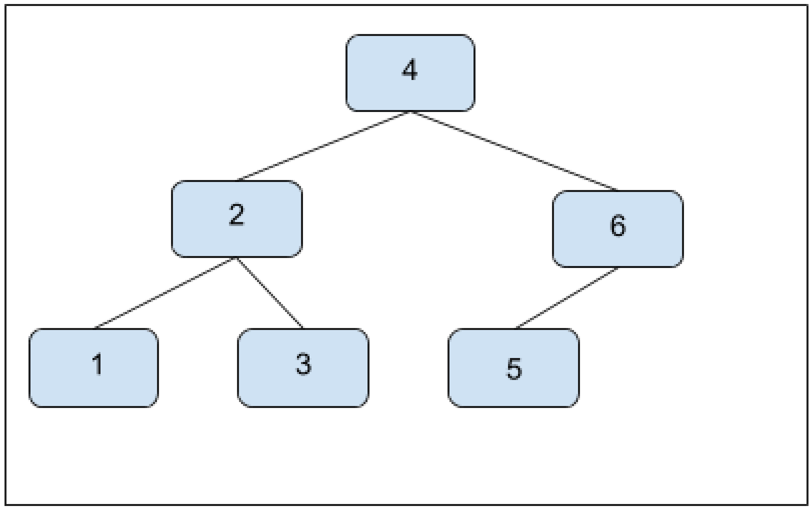
Draw a min-heap for the following numbers [5, 6, 3, 1, 8, 9]

In sequence perform the following operations:

1. Delete the root node
2. Delete the left-most leaf node
3. Delete the rightmost non-leaf node

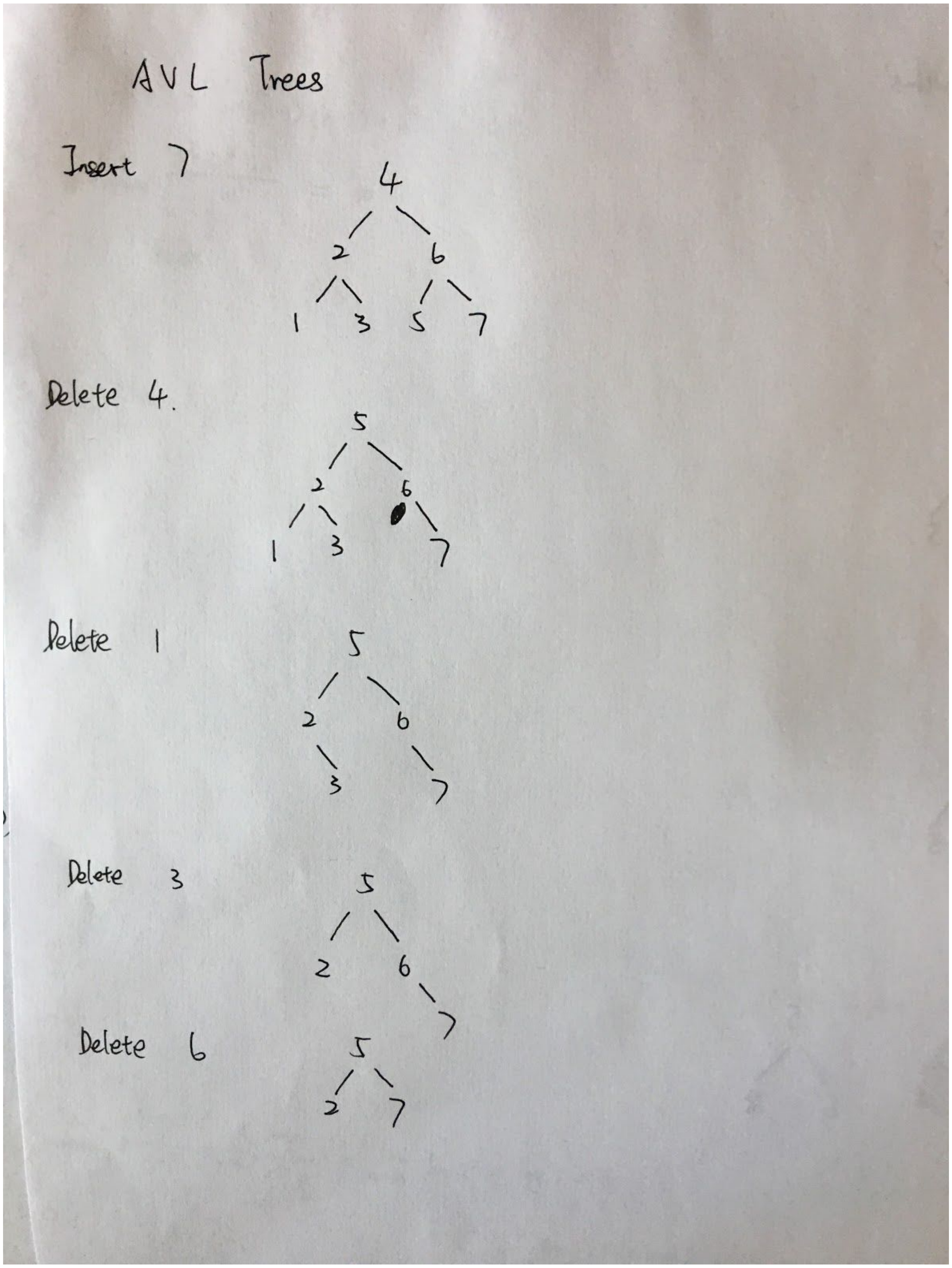


Problem e1.7 AVL Trees



Convert the tree above into an AVL tree, then show the results of the following sequential operations

Insert 7
Delete 4
Delete 1
Delete 3
Delete 6



Complexity Definitions and Properties

Prove or disprove the following claims:

A) Claim $f(n) = n! \in O(n^2)$
   Pf:

B) Claim $f(n) = n! \in O(n^n)$
Pf:

C) Claim $f(n) = 2^{log(2^n)} + 2^{2n} \in O(2^n)$
Pf:

D) Show $O(n)$ *is a proper subset of* $O(n \log n)$
Pf:

E) Show $O(n \log n)$ *is a proper subset of* $O(n^2)$
Pf:

Answers here:

Complexity Proofs

A) $n! \geq n(n-1)(n-2) = n^3 - 3n^2 + 2n$

$\lim\limits_{n \to \infty} \dfrac{n^3 - 3n^2 + 2n}{n^2} = \infty$ So $f(n) = n! \notin O(n^2)$

False

B) $\lim\limits_{n \to \infty} \dfrac{n!}{n^n}$ converges by ratio Test.

So $f(n) = n! \in O(n^n)$ True

C)

c)

$$2^{\log(2^n)} + 2^{2n} \geq 2^{2n}$$

$$\lim_{n\to\infty} \frac{2^{2n}}{2^n} = 2^n = \infty \qquad \text{so} \quad f(n) = 2^{\log(2^n)} + 2^{2n} \notin O(2^n)$$

D)

$$\lim_{n\to\infty} \frac{n}{n\log n} = \lim_{n\to\infty} \frac{1}{\log n} = 0$$

so $O(n)$ is a proper subset of $O(n\log n)$

E)

$$\lim_{n\to\infty} \frac{n\log n}{n^2} = \lim_{n\to\infty} \frac{\log n}{n} = \lim_{n\to\infty} \log n^{\frac{1}{n}} = \log 1 = 0$$

so $O(n\log n)$ is a proper subset of $O(n^2)$