Problem Set 2

Graded

Student

Anton Melnychuk

Total Points

99 / 100 pts

Question 1

Q0: Your Information

1 / 1 pt

✔ **– 0 pts** Correct

Question 2

Q1: Number Game

14 / 14 pts

✔ **+ 5 pts** Correct Algorithm

✔ **+ 5 pts** Proof of Correctness

✔ **+ 4 pts** Correct running time analysis

Question 3

Q2: Road Trip with Refueling — **29** / 30 pts

3.1 **Q2.1** — **10** / 10 pts

- ✔ **+ 5 pts** Correct Algorithm
- ✔ **+ 2.5 pts** Correct proof of correctness
- ✔ **+ 2.5 pts** Correct running time analysis

1. Comment: Is this log(k)? You seem to have log(n) above?
2. We only need to check the distance to one vertex: v

3.2 **Q2.2** — **9** / 10 pts

- ✔ **+ 5 pts** Correct Algorithm
- ✔ **+ 2.5 pts** Correct Running Time Analysis
- ✔ **+ 2.5 pts** Proof of Correctness
- ✔ **– 1 pt** Error

3. Good practice to not reuse variables (e.g., v) already defined in the problem statement
4. The LHS is O(n^2+k(m+n)log(n)) and cannot be simplified to O(k(m+n)log(n)) unless $k \geq \Omega(n/\log(n))$
5. Distance tables only need to be computed for O(k) vertices; once for each call of Dijkstra

3.3 **Q2.3** — **10** / 10 pts

- ✔ **+ 5 pts** Correct Algorithm
- ✔ **+ 2.5 pts** Correct proof of correctness
- ✔ **+ 2.5 pts** Correct running time analysis

6. Same comment as for part 2.

Question 4

Q3: Maximum Weight Connected Subgraph

Resolved **20** / 20 pts

✔ **– 0 pts** Correct

7 O(m)

8 I'm not sure what this means, you've already had the maximum spanning tree, why you need to recheck the connectivity? I deduct points here but feel free to make regrade request explaining your ideas.

Regrade Request **Submitted on: Oct 21**

Hello, thank you so much for leaving the comment! I'd like to kindly clarify that the section in question is an overview of how to implement Prim's Algorithm, which was followed by the content presented in our class. I wasn't entirely sure how much rigorous I need to be in the algorithm explanation, and I apologize for any confusion this may have caused. Wishing you a wonderful evening!
Let me know if there are other issues. Thank you so much!

oh i see, points back

Reviewed on: Oct 23

Question 5

Q4: Finding Array Element

15 / 15 pts

✔ **– 0 pts** Correct

Question 6

Q5: A Faster Algorithm to Multiply Integers **20** / 20 pts

6.1 **Q5.1** **10** / 10 pts

✔ **– 0 pts** Correct

9 To use the Master Theorem, you must show or claim that T(1) -- the base case -- runs in constant time.

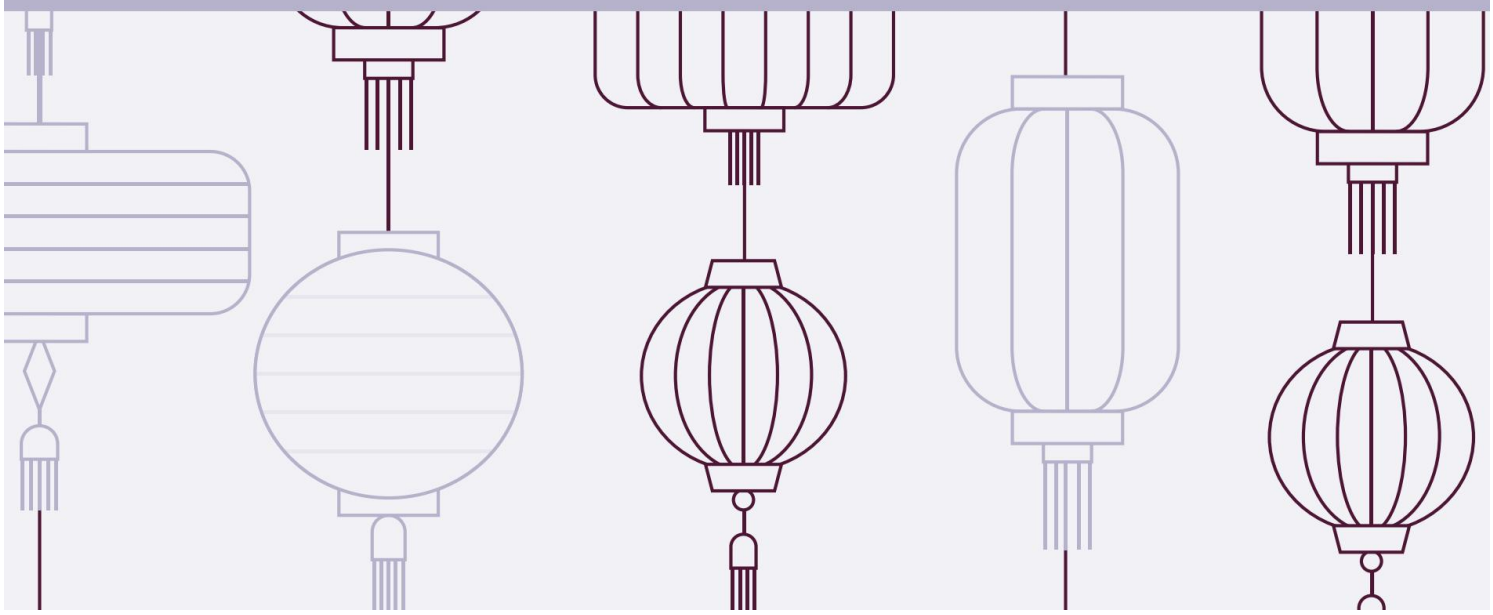
6.2 **Q5.2** **10** / 10 pts

✔ **– 0 pts** Correct

10 Great!

No questions assigned to the following page.

Anton Melnychuk
CPSC 365 ※2



Question assigned to the following page: 1

Problem #0

(a) Anton Melnychuk

(b) am 3785.

(c) ULAs: Brian, Prastik, Caleb

(d) I've followed the academic integrity and collaboration policy as written above.

(e) 20h.

Question assigned to the following page: 2

Problem #1.

Step #1

1) Initialize an array can_win[n+1] size of n+1 b/c $0 \leq m < n$, +1 for inclusive n (root).

2) Similarly to dynamic programming, set all values in can_win[] table to 0. Moving from given n to m, set initial element can_win[n] = 1 to be reachable

Step #2.

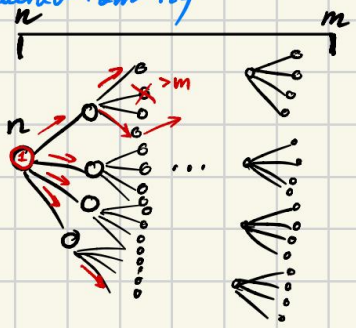
1) Let k denotes an element in the table can_won[]. Then, can_win[k] says if it is possible to reach number k from n.

2) Without lost of generality, lets move through numbers in decreasing order from n. For each $k \leq n$ apply:

If can_win[k] = 1: (skip all k=0, that cannot be reached from n.)

(i) update each can_win[k'] = 1:

- can_win[$\frac{1}{3}$k] = 1
- can_win[k - $\frac{1}{3}$k] = 1) (1)
- can_win[$\sqrt{k}$] = 1
- can_win[k - $\sqrt{k}$] = 1) (2)



if one is within bounds [n, m].

Step #3.

Continue iterating over all k until processed all possible numbers / combinations of all $\frac{1}{3}$k, $\frac{2}{3}$k, $\sqrt{k}$, k-$\sqrt{k}$ cases.

Step #4

After the loop has finished, can_win[m] should tell if we can reach number m from n. (check if == 1).

Proof: Similarly to brute-forcing algorithm, for each k we have 4 other cases we iterate over again in the decreasing order as long as the result is in the boundaries of "looking-for" number m and root n. The decreasing order allows algorithm to run through all possible outcomes without repeating itself further on, while the boundaries keep the complexity in the range of given numbers.

Time Complexity: b/c we iterate over each possible member one at most n-m times, where $0 \leq m < n \Rightarrow O(n)$. (as any option would lead k to k' s.t. $|k'| < k$ (log sqrt/divis. dec))

Question assigned to the following page: 3.1

Problem #2

Let $V = \{c_1, c_2, c_3 \dots c_n\}$ – set of cities, along with highways (edges) E.
Denote such graph as $G = (V, E)$.

Part #1.

Step #1. Initialize distances:
$d(s) = 0$, $d(v) = \infty$ for all $v \in V; v \neq s$
(create a table).

Step #2.
1) Create an a binary heap as an priority queue.
   (1) list of vertexies $V = \{u, v_1, v_2, \dots, v\}$ / all cities
   (2) make a min heap tree / sort them $O(n \log n)$.
2) While Q is not empty
   (a) Extract u (smallest) vertiex
   (b) For each neighbor of n_v (notation) of this v: $(v, n_v) \in E$
      (i) Calculate tentative distance and check / update:
         if $d[u] + w(u, v) > d[v]$
         update $d[v]$ to a new value
         $d[u] + w(u, v)$

Dijkstra's Algorithm

Step #3

1) Using table defined on the step #1 check if $d[v'] \leq L$, where v' is a final city-destination, which is the answer to this problem.

Proof: The graph is weighted, connected, and undirected to found if there exist a path s.t. a car can reach city v from u, it's enough to check if the shortest path is $\leq L$ (max miles without refuelling) instead of brute-forcing all possible routes. Therefore, as proven in class (Dijkstra's), I use same concept on the same parameters to solve this problem.

Time Complexity:

- Dijkstras Algorithm: $O((m+n)\log n)$ (From the class).
- Checking distance: $O(1)$ per node [2] $\Rightarrow O(n)$ in total

Thus, time complexity is $O((m+n)\log n)$ [1].

Question assigned to the following page: 3.2

Part*2

Let's denote the set of city-gass-stations as $S = \{c_{k+1}, c_{k+2}, \dots c_k\} \in V$ for k cities (by problem def.) and graph of all cities $G = (V, E)$, analogically.

Let X be a set of (all) city-gass-stations at some layer, where layer is bounded by the amount of times car travels btw\ any two gass-stations (call it k). Set $X = \{s\}$ for the start, where s is a source vertex (city) we start with. Now let X' be a set of (all) city-gass-stations reachable in the next layer. Set $X' = \{\emptyset\}$, then:

Algorithm:

Initialize global boolean table for each vertex: $e(s) = true$, $e(v) = false$ for all gass-stations $v \in S$; $v \neq s$ — if explored by Dijkstra's Algo.

s.t. $e(v) == 0$ *

For each vertex in X (layer) run Dijkstra's Algorithm described in the Part*1 and update $e(v) = 1$ (explored).

for each found v in the table of shortest distances (look def. part*2) s.t.

$$(d[v] \leq L) \wedge (e[v] == false) \wedge (v \in S \in V)$$ (denote as reach(v))

(1) check if destination → return True

Otherwise, (2) append this vertex to X' (might have copies) *

If no destination found => run over the Algorithm with a new:

$$X = X' \quad \text{and} \quad X' = \emptyset$$

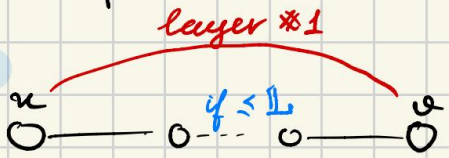
Iterations should end either when all vertices that are gass-stations (S) are "visited" (false) OR if the destination is found.

Question assigned to the following page: 3.2

Proof:

By Induction:
Let k be the amount of times ever travels btw\ any two gass-stations. Then L_k gives the set of all vertices we can reach by refuelling k times.

Base Case: $k = 1$:

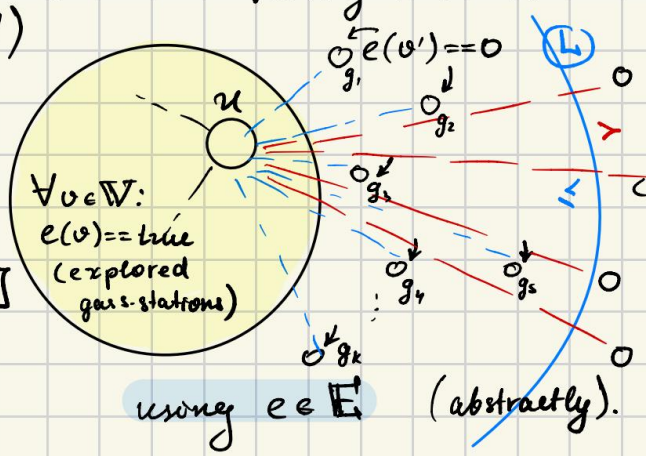


For this case recall part #1 ✓

Induction Hyp.:
Assume that after some layer k, the vertices/gass stations found by Dijkstras are indeed:

(1) not-explored (2) gass-stations (3) $d[v] \leq L$ (reachable).

+ (4) There (strong induction) exist path from u to each vertex that is "explored." (note the graph on the right)



Induction Step:
[WTS: My algorithm will correctly say if there is a path to the next unexplored gass-stations vertices in $k+1$]

Let's denote "reached" gass stations as $g_1, g_2, g_3, \ldots g_k$, where $g \in S \wedge e(g) = 0 \wedge d[g] \leq L$.

To find if there exists any path from any g to the next unexplored vertex, it is enough to find the shortes weighted path and check if it's $\leq L$. Otherwise, all other pathes greater than that are definitely not in the range of L miles (by min. prop.)
Therefore, using Dijkstras for connected/weighted/undirected, we garantee that this condition holds correctly (by proof in the class). $\therefore X' = \text{reach}(\forall v \in X)$.
we reach a new layer!

Next [WTS: The algorithm halts]

B/c the graph is connected, we know that there exists some combination of vertices to reach v from u, where v - destination.

Question assigned to the following page: 3.2

By direct proof, b/c the algorithm brute-force each "reachable" point ($\leq L$) gass-station, and we immediately denote it as "explored", the algorithm correctly finds next city-gass-stations just excluding ones already used before. Running Dijkstras on the next gass-stations it exhausts cities to be explored next in G.

Thus, by set-up of my algorithm every time we find vertices, running Dijkstras Algorithm, that is our destination and still satisfies all other conditions, we halts at true!

Otherwise, if ran out of all vertices => halts at false!
(could be reached)

Time Complexity:

Dijkstras: $O((m+n)\log(n))$.
Go through each vertices: $O(k)$
] => $O(k(m+n)\log n)$. run Dijkstra for each vertex

Build a distance table => $O(n)$. per each $n \in V$ (5)
Build an "explored cities" table => $O(n)$. once.

Thus, $nO(n) + O(k(m+n)\log n) = O(k(m+n)\log n)$, where k is a number of gass-stations. and $m \in E$. (4)

Question assigned to the following page: 3.3

Part*3

Algorithm:

(1) Consider algorithm from the part*2 problem *2.
Let's initialize global variable $c = 0$ to be the cost required to pursue some path built by the prev. algorithm to all reachable city-gass-stations from the source S.

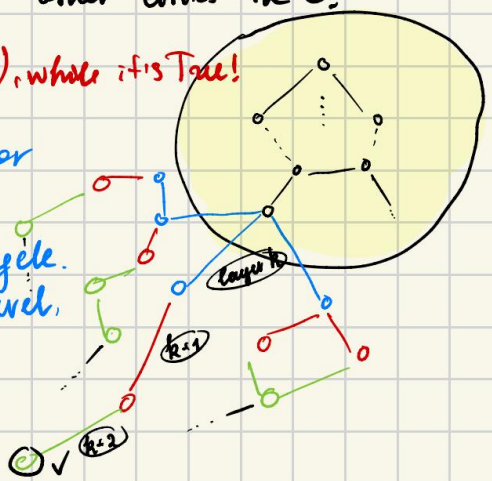
(2) Running Algorithm from the part*2, every time we create set $c += 1$. to keep count of k or amount of layers.

(3.) When Algorithm halts on the destination:
if $c \leq D$, return true
if $c > D$, return false
Otherwise, return false (ran out of vertexies)

Proof: Consider the proof in part*2. In fact, it's a solution for part*3.
[WTS: The first time algorithm halts on destination is found by the shortest / the greediest possible way in G (has least amount of city-gass-stations among all other pathes)] => C_{min}.

By proof from the part*2, we know that the algorithm finds all reachable city-gass-stations optomaly among all other cities in G.

By contrapositive, let it halts (from part*2), but says False ($c > D$), while it's True!
b/c before inductively iterating algorithm for the new found vertexies, the algorithm makes sure we're finished the previous cycle. and only then increased c by 1$ to the next level.
and
b/c each level is made from Dijkstra's algorithm distances table, so its proof can guarantee that there exists at least 1 possible way of development (by shortest) while brute-forcing it for all city-gass-stations vertexes, then:



Question assigned to the following page: 3.3

If no destination is found on k's level, then for any $k' < k$, there should not exist any other ways to reach destination with a lower cost $c = k$ as we already in-order of layers brute-forced them; and if shortest path somewhere larger than L miles, than any other paths longer than that are also $> L$ (by min. def.) Therefore, if algorithm halts on the destination, the first should be one that had min amount of city-gass stations. ✓

Checking: if $c(\text{greadest}) \leq D$:
return true!
Otherwise, return false.

greadest — uses as most as poss miles s.t. $\leq L$, so reaches faster.

Q.E.D.

Time Complexity:

Dijkstras: $O((m+n)\log(n))$.
Go through each vertiey: $O(k)$
$\Rightarrow O(k(m+n)\log n)$. run Dijkstra for each vertex

Build a distance table $\Rightarrow O(n)$. per each $n \in V$.
Build an "explored cities" table $\Rightarrow O(n)$. once.

Append to X/X', update $X/X' \Rightarrow O(k)$.

Thus, $O(k(m+n)\log n)$, where k is a number of gass-stations. and $m \in E$.

Question assigned to the following page: 4

Problem #3

Given:

Let $G=(V,E)$ - connected, undirected, and weighted graph, where each $e \in E$, with $w(e) \in \mathbb{R}$. Find MWCS-? Denote as F of (G,w) is a subset of edges $F' \in E$ that max the total weight. $w(F) := \sum_{e \in F'} w(e)$.

Algorithm:

(1) Let $E^- = \{\forall e \in E.\ w(e) = w(e)(-1)\}$ and $G^- = (V, E^-)$. [7] $[O(n)]$.

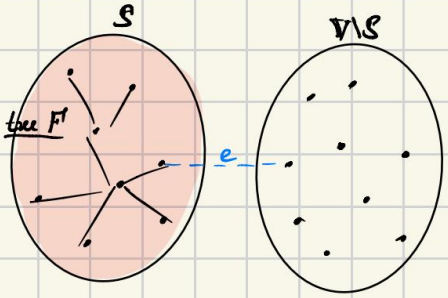
(2) Run Prim's Algorithm to determin the MST in G^- st. all vertices are connected with minimum cost (any $e<0$ that heave been negated to the positive values).

(i) Start from $F' = \{s\}$ for any root node $s \in V$. [8]

(ii) In each iteration:

- Find the cheapest edge e connecting S and $V \setminus S$. i.e. find a vertex $v \notin S$ with min cost:

$$cost(v) := \min_{\substack{u \in S \\ (u,v) \in E'}} e(u,v)$$



- Add v to S, and add $e=(u,v)$ that achieves the min to the tree F'.

Thus, it still remains a tree — only one connected component (prop. of Prim's Algo.)

(4) Add all positive vertices in G to subgraph F' to maximize the total weight. b/c any $e \in E' : e > 0$ would maximize the total weight:

$$w(F') := \sum_{e \in F'} w(e) \text{ (by add property).}$$

(5) Return Subgraph $F' \leftrightarrow$ MWCS.

Question assigned to the following page: 4

Proof:

(1) During the Prims execution for the MST connecting all $v \in V$, we negated all weights to a new graph G^- s.t. negative weights are used as least as possible.

(2) In each iteration, Prim's algorithm adds the edge $e = (u, v)$ where $v \notin S$ minimizes $\text{cost}(v) = \min_{u \in S} c(u, v)$.

This is the cheapest edge connecting S and $V \setminus S$.
By Cut Property, this edge e must be in any minimum spanning tree. (proved in class) Therefore, along the iterates of Prim's, the tree is a subset of any minimum spanning tree

Prims algorithm ends when we have added all vertexes; then the final tree is a spanning tree, using as least as possible max weights, which are negated.

(3) Now, having built connected subgraph, using min cost expenses, we add all possitive values to max the total weight (by add. prop.)

Q.E.D.

Time Complexity.

By definition of Prim's Algorithm (proved in class), time complexity of part (2) is $O([m+n]\log n)$.

Complexity for negating all weights is $O(n)$ b/c we need to run through each edge and negate it.

Same for finding all positive values and adding them to the graph to maximize it. $n \times O(1) = O(n)$.

$\therefore O((m+n)\log n) + O(n) + O(n) = O((m+n)\log n)$.

Question assigned to the following page: 5

Problem *4

Given array $A = \{x_1, x_2 \dots x_n \mid x \in \mathbb{R}\}$, $|A| = n$. Assume the entries of A are sorted from smallest to largest. Find Algor. that determine if exists $i \in \{1, \dots, n\}$ s.t. $A[i] = i$. (Even though $i \neq 1$, algorithm halts in false if it did not find the answer.)

Algorithm:

Step *1. Initialize two pointers.
Let's denote them as l and r, to the first and last indices of number array A.

Step *2 While $l \leq r$:

1) Calculate the mid-index:
$mid = (l + r) // 2$

2) Compare $A[mid]$ with mid:
- if $A[mid] == mid$:
  return mid.
- if $A[mid] > mid$:
  $r = mid - 1$ since the array A is sorted and for current index mid, values in the list are greater than the index we look for.
- if $A[mid] < mid$:
  $l = mid + 1$ analogically.

3) If the loop terminates without finding an index i s.t. $A[i] = i$, return false. Otherwise, true.

Question assigned to the following page: 5

Proof: By Induction, let's prove that the algorithm provided above correctly returns index of number if number $\in A[i]$, $i = \{1, 2, 3 \dots n\}$ by def; or false if not.

Base Case ($n = 1$): the algorithm checks if $A[i] = i$ and clearly returns if this true or false, where i is $r - l + 1$ (b/c we start with one)

Induction Hypothesis:

Assume that for range of size $< k$ ($k \geq 1$), my algorithm halts (return true if $x : A[x] = x$ is there, otherwise it returns false).

By strong induction, WTS: my algorithm in range of size k elements returns true if such x also exists there.

Induction Step:

In k range, my algorithm will execute recursive case that compares $A[mid]$ to mid itself

Case #1: $A[mid] < mid$, $mid = l + r/2$.

- Since A sorted, mid must be btw\ position mid+1 and r.
- ... which is serched recursively.
- ... which also returns true/false by induction hypothesis.

Case #2: $A[mid] > mid$

- By similar reasoning, mid must be btw\ l and mid-1, which are correctly searched recursively.

Case #3: $A[mid] = mid$

- Algorithm returns true, which is correct b/c mid is clearly in array range and set conditions.

Mid point has to be btw\ l and r for recursion to work on smaller range, otherwise its computation never affected proof

Time Complexity: In each step reduce size by $/2 \Rightarrow O(\log n)$ (from class) similarly to Binary Search as functionality is the same, the case is differ.

Question assigned to the following page: 6.1

Problem #5.

Assume $n = k^m$ for some integer $m \geq 1$, so n/k is an integer. Inputs: Two n-bit integers, M and N, and k - const.

Part #1

Step #1 Decode both M and N into n/k chunks.
Step #2. Recursively apply the FastMult(M_i, N_i, k) algorithm for each pair of corresponding k-bit chunks. This continues until the chunks are small enough to be processed directly s.t. they become n/k integers.
Step #3. Combine the results to reconstruct the final product of M and N. (Regardless which M_i & N_i)

Algorithm:

FastMult (M, N, k): (used here) ← given FastMult.
if k == 1 return M*N.

(split equaly into k chunks)
$M_{chunks}, N_{chunks} \leftarrow$ Split (M, k), Split (N, k).

Multiplications: (given at most $2k-1$) e/c of ① starting point
For each i from 0 to $(2k-1)-1$!
$M_i = M_{chunks}[i]$
$N_i = N_{chunks}[i]$
⇒ result += FastMult (M_i, N_i, k) << $(i * n/k)$.
(recursively decide) (shift left.) (used for multiplic. again).

Addition: (given at most k^3)
For each i from 0 to k-1
M_{sum} = Sum ($M_{chunks}[i(2k-1) : (i+1)(2k-1)]$)
N_{sum} = Sum ($N_{chunks}[i(2k-1) : (i+1)(2k-1)]$)
result += FastMult (M_{sum}, N_{sum}, k) << $(i * n/k)$

return result.

Questions assigned to the following page: 6.1 and 6.2

Time Complexity: By Master Theorem: 9

$r = \frac{2k-1}{k} = 2 - \frac{1}{k} > 1$, for all $k > 1$ (by def.) [not $k \neq 1$ from ed #51]

Thus, $T(n) = (2k-1)T(n/k) + \mathcal{O}(n) \in \mathcal{O}(n^{\log_k 2k-1})$.

[b/c k^3-const.], $m \neq 1$. b/c $r > 1$.

Part #2 No. The equation is wrong.

$k = n$ [WTS: $\mathcal{O}(n^{\log_n(2n-1)}) \stackrel{?}{=} \mathcal{O}(n)$]

$\lim_{n \to \infty} \log_n(2n-1) = 1$.

$T(n) = aT(n/n) + \mathcal{O}(n^4) =$

$= (2n-1)T(n/n) + \mathcal{O}(n^4)$.

$= (2n-1)T(1) + \mathcal{O}(n^4)$

Thus, $\mathcal{O}(n) + \mathcal{O}(n^4) \in \mathcal{O}(n^4)$. 10

Q.E.D.