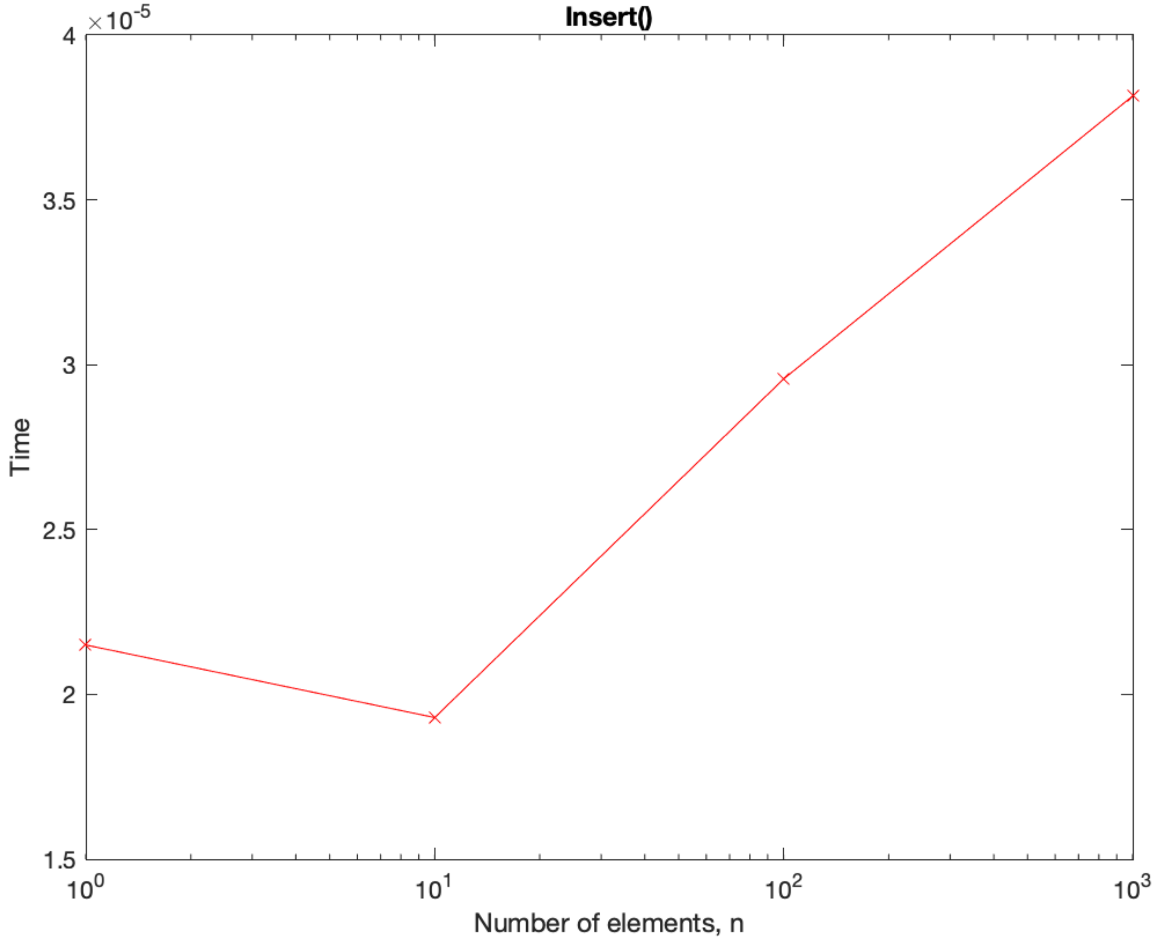
William Wijaya
3035992465

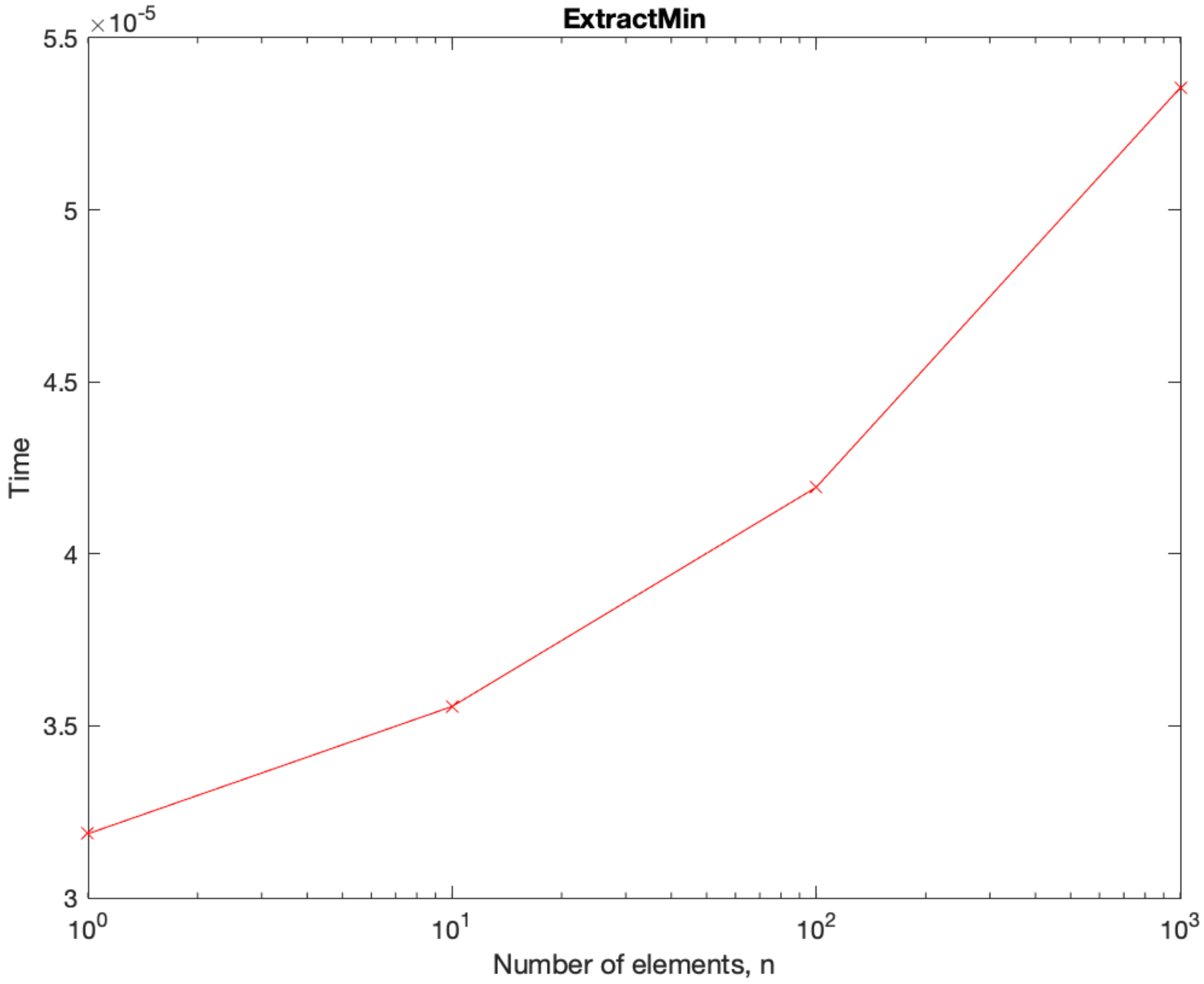
Lab #5
Due: Wednesday 4/28 at 11:59pm

Lab 5: Dijkstra's Shortest Path Algorithm



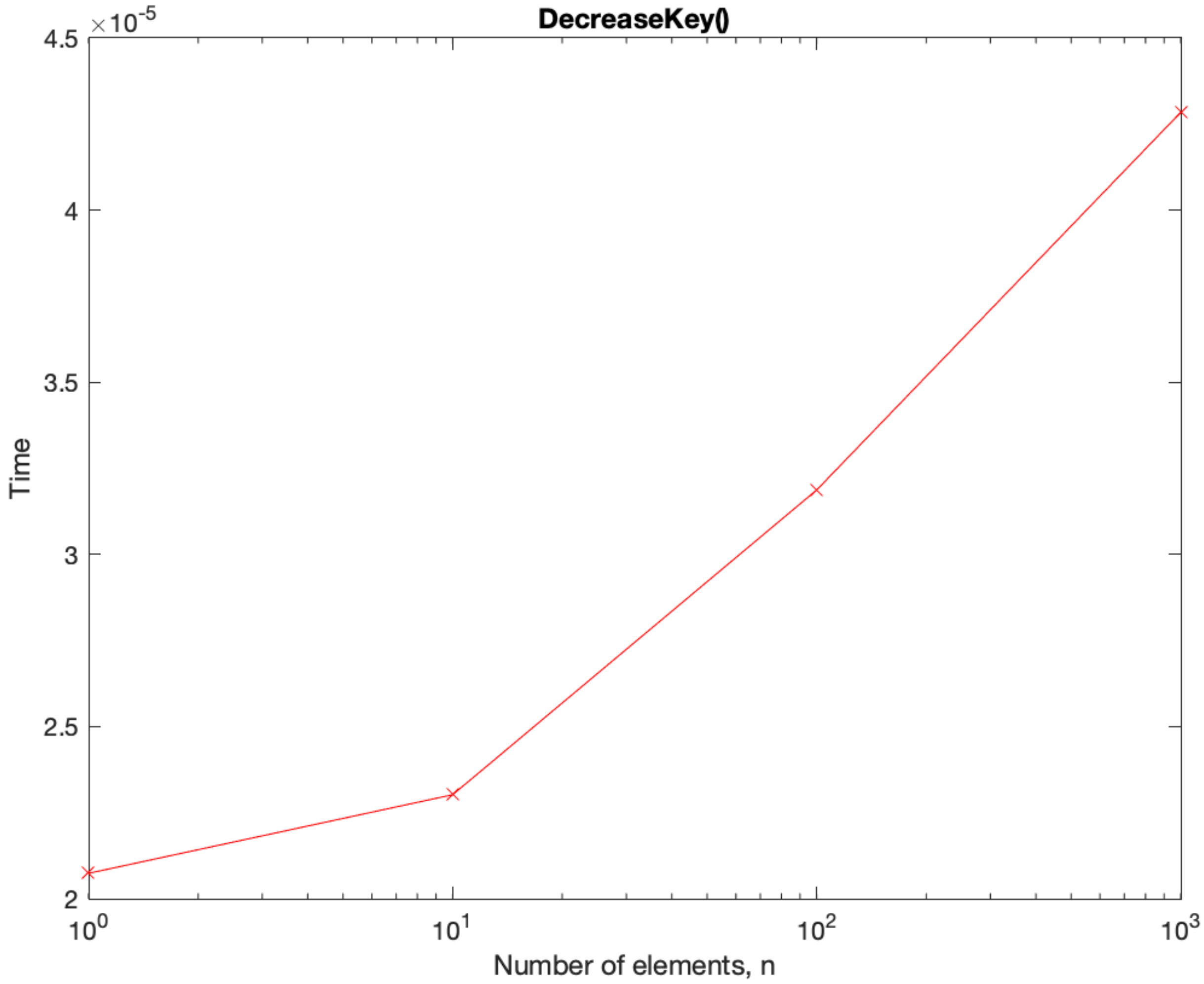
Comment:

The x axis is scaled based on log(n). We can see that from n=10 to n=10^3, the graph is almost linear. However, from n=10^0 until n=10^, the time for n=10^0 is higher than n=10^1, which indicates that in that range the graph is not linear. This happens because we have few elements on that range. So, in that range of n, the process is basically instantaneous. We are doing so much checking in very few operations. We can't really see how much time the operation does take. But, in the context of complexity, we don't really care about lower range of n, we do care about n as n goes to infinity. We can see that as soon as n increases, we see a linear relationship in log plot. Hence, we can verify that insert command is indeed linear in log(n).



Comment:

We can see that the graph for log(n) in ExtractMin command is almost linear. Thus, we can verify that the complexity ExtractMin command is linear for log(n) graph.



Comment:

We can see that the graph for log(n) in DecreaseKey command is almost linear. In the lower range of n, the graph is a little bit nonlinear, but this can happen because due to the device performance as previously stated in my previous comment on insert command. We really care when n goes to infinity. For higher range of n, the graph is linear. Thus, we can verify that the complexity ExtractMin command is linear for log(n) graph.

	dist														
graph 1	0	3	9	2	11	7	4								
graph 2	0	1	3	Inf	Inf	Inf									
graph 3	0	Inf	1	5											
graph 4	0	84	11	32	58										
graph 5	0	663	466	995	319	483	496	376	382	528					
graph 6	0	277	556	458	245	9	104	780	642	346	36	359	314	138	215

	prev														
graph 1	0	1	2	1	6	4	4								
graph 2	0	1	2	0	0	0									
graph 3	0	0	1	3											
graph 4	0	4	1	1	1										
graph 5	0	6	8	5	1	3	1	1	5	3					
graph 6	0	11	13	11	6	1	11	11	5	13	6	6	15	11	11

Code:

William_Wijaya_lab5.m

```
%% This section is for problem 1 in lab 5
%% Insert pq command
clear all,clc,close all

a=0:3;
n=10.^a;


for R=1:500 %Repeat the process "R" times
    for i=0:3 % for n=10^i
    pq=PriorityQueue();
    % Inserting "n" element in the priority queue starting from priority 2
        for j=2:n(i+1)+1
        pq.Insert(j-1,j);
        end
        start=tic; % Start timer
        pq.Insert(j,1); %Inserting a first priority element into priority queue
        stop=toc(start); % Stop timer
        tEnd_Insert(i+1,R)=stop; % Record time
        %clear pq % Clear priority queue and start from the beginning
     end
end

tEnd_Insert=mean(tEnd_Insert,2);
semilogx(n,tEnd_Insert,'r-x')
xlabel('Number of elements, n')
ylabel('Time')
title('Insert()')

%% ExtractMin command
clear all,clc,close all
a=0:3;
n=10.^a;
```

```
for R=1:500
    for i=0:3
        pq=PriorityQueue();
        for j=1:n(i+1)
            pq.Insert(j,j);
        end
        start=tic;
        pq.ExtractMin();
        stop=toc(start);
        tEnd_Insert(i+1,R)=stop;
    end
end

tEnd_Insert=mean(tEnd_Insert,2)
semilogx(n,tEnd_Insert,'r-x')
xlabel('Number of elements, n')
ylabel('Time')
title('ExtractMin()')

%% Decrease key command
clear all,clc,close all
a=0:3;
n=10.^a;

for R=1:500
    for i=0:3
        pq=PriorityQueue;
            for j=2:n(i+1)+1
            pq.Insert(j-1,j);
            end
        start=tic;
        pq.DecreaseKey(j-1,1);
        stop=toc(start);
        tEnd_Insert(i+1,R)=stop;
        clear pq
    end
end

tEnd_Insert=mean(tEnd_Insert,2)
semilogx(n,tEnd_Insert,'r-x')
```

```
xlabel('Number of elements, n')
ylabel('Time')
title('DecreaseKey()')

fprintf("\nDone!\n")
```

myDijkstra.m

```
% Dijkstra's algorithm for fidnings the shortest path from
% an origin node to all otehr nodes. The function takes an edgelist
% of the form [node1 node2 edgeCost].

function [dist,prev] = myDijkstra(edges, origin)
%your code here

% dist[source]<- 0
dist=[];
dist(origin)=0;

% create vertex priority queue Q
pq=PriorityQueue;

graph2=edges(:,1:2);
uni=unique(reshape(graph2,[1,numel(graph2)]));
lgraph=numel(uni); %how many nodes in graph.edges

% for each vertex v in a graph where v is not source
dist(2:lgraph)=inf;
prev=zeros(1,lgraph);

% Q.add_with_priority(v,dist[v])

for i=1:lgraph
    pq.Insert(i,dist(i));
end

% while loop when Q is not empty
```

```matlab
for z=1:lgraph
    u=pq.ExtractMin();
    lfind=find(edges(:,1)==u)'; %finding the first column of graph.edges=u
    edgesN=edges([lfind],:); %Simplifying graph.edges that contains u
    edgesN2=edgesN(:,1:2); %Simplified graph.edges two first columns
    edgesN3=edgesN(:,3); %Simplified graph.edges for the third column
    neighbor=reshape(edgesN2,[1,numel(edgesN2)]);
    neighbor(find(neighbor==u))=[];
    neighbor=sort(neighbor); %finding the neighboor of u in ascending orders

    for j=1:length(neighbor)
        lengthh=edgesN3(find(edgesN2(:,2)==neighbor(j)));
        alt=dist(u)+lengthh;
        if alt<dist(neighbor(j))
            dist(neighbor(j))=alt;
            prev(neighbor(j))=u;
            pq.DecreaseKey(neighbor(j),alt)
        end
    end
end
dist=dist;
prev=prev;
end
```