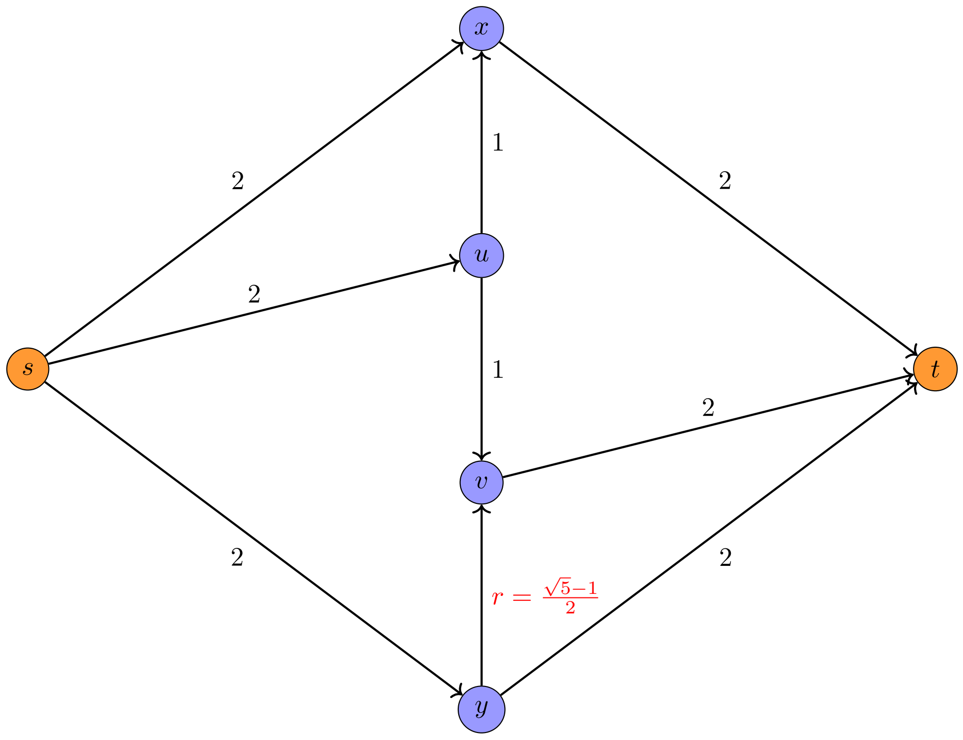
CS345: Design and Analysis of Algorithms

Assignment 4

Submitted by: Aditi Khandelia, 220061

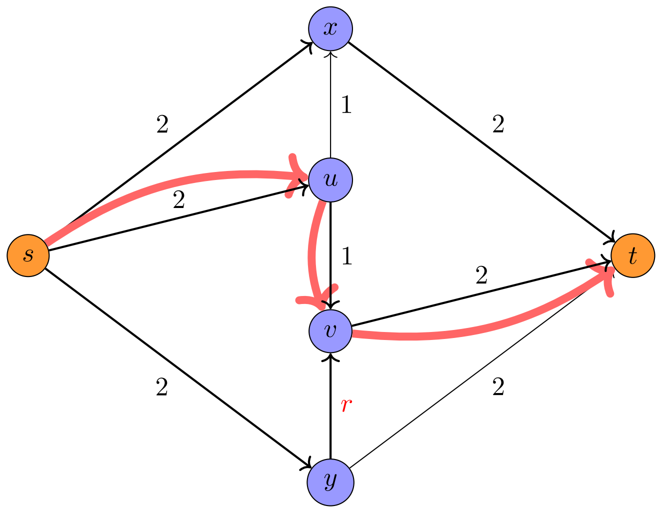
Question 1.
Recall the example of a flow network with 6 nodes where the Ford-Fulkerson algorithm may never terminate. Furthermore, even in the asymptotic sense, the flow computed will be less than the maximum flow possible. Please provide a comprehensive analysis of this scenario, thereby showing how the algorithm may run forever.

Answer.
Following is the graph G with 6 nodes and 9 edges on which Ford-Fulkerson can be proven to never terminate:

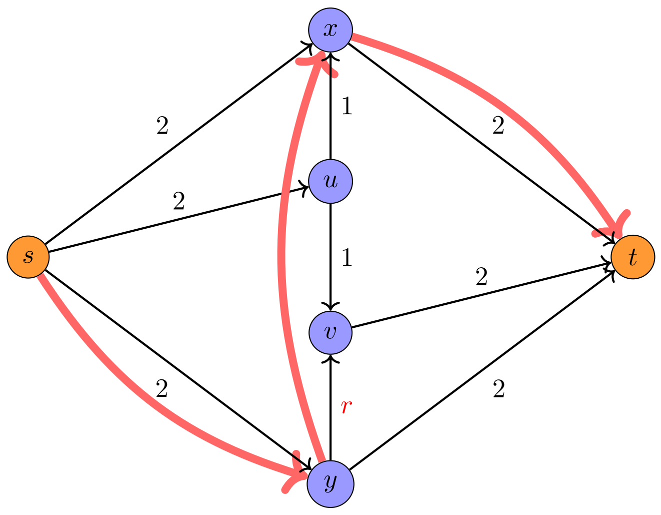


We'll define four paths on this graph :

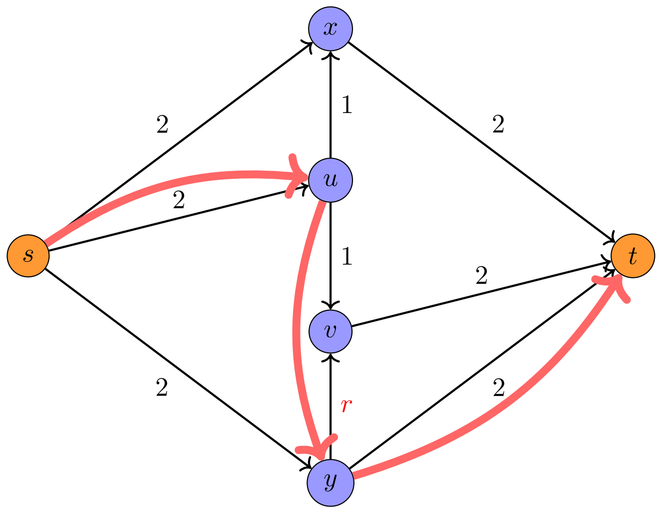
1. Path $P0$:



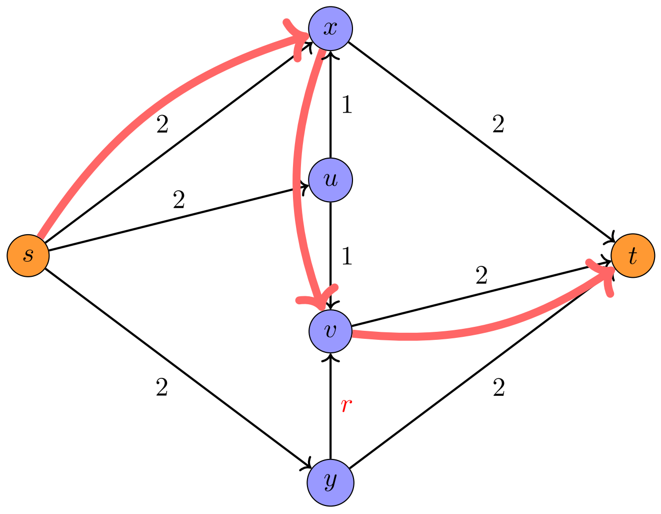
2. Path $P1$:



3. Path $P2$:



4. Path $P3$:



Ford-Fulkerson algorithm doesn't specify any metric for choosing an augmentation path in each step, as long as all the edges of the path lie in the residual graph at the stage. Thus, choosing any sequence of paths in the residual graph which leads to non-termination will prove that it there is a possibility of non-termination in Ford-Fulkerson algorithm.
Thus, we'll use the following sequence of paths on the graph G : $P0, P1, P2, P1, P3, P1, P2, P1, P3, \ldots$

Proof by Principle of Mathematical Induction

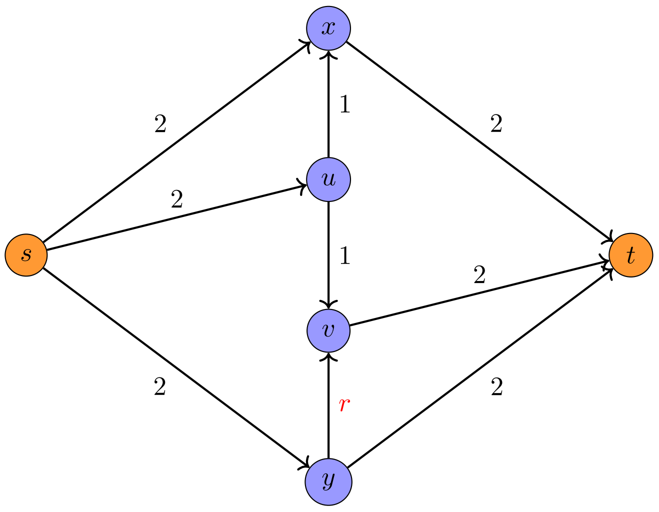
Claim

After k^{th} application of the sequence $P1, P2, P1, P3$, following will be the residual capacities along the edges :

1. $(s,y) = 2 - (r + r^2 + \cdots + r^{2k-1} + r^{2k})$
2. $(s,u) = 2 - (r + r^3 + \cdots + r^{2k-1})$
3. $(s,x) = 2 - (r^2 + r^4 + \cdots + r^{2k})$
4. $(x,t) = 2 - (r + r^2 + \cdots + r^{2k-1} + r^{2k})$
5. $(v,t) = 2 - (r^2 + r^4 + \cdots + r^{2k})$
6. $(y,t) = 2 - (r + r^3 + \cdots + r^{2k-1})$
7. $(y,v) = r^{2k+1}$
8. $(u,v) = 0$
9. $(u,x) = r^{2k}$

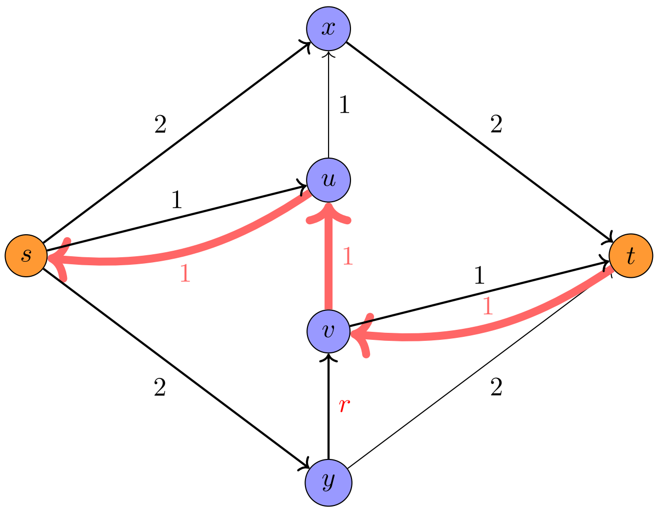
Base Case
We prove the base case by simulating the first sequence on the graph G. Following is the residual graph G' on each step of executing the given sequence on G :

1. Initial :



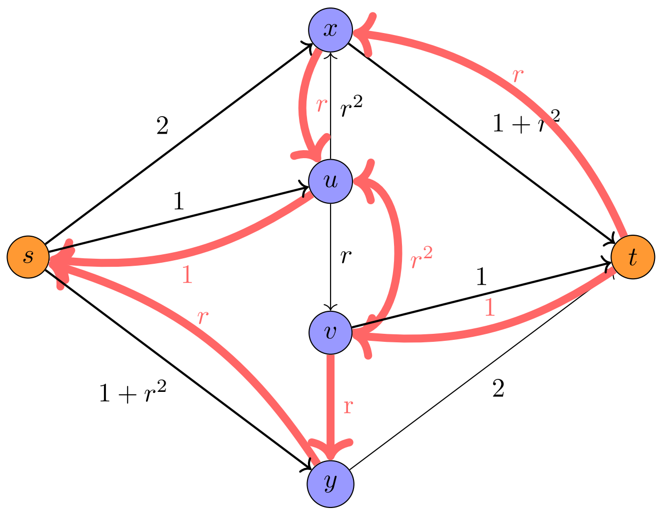
All the edges of the path $P0$ are present in the residual graph, allowing us to simulate it next.

2. After path $P0$:



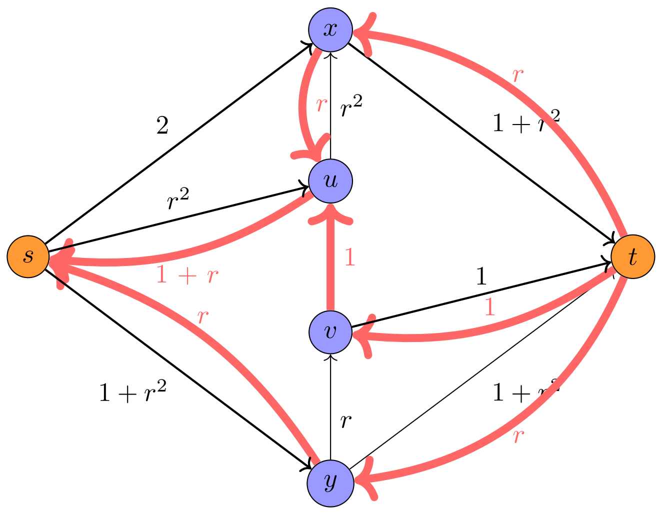
All the edges of the path $P1$ are present in the residual graph, allowing us to simulate it next.

3. After path $P1$:



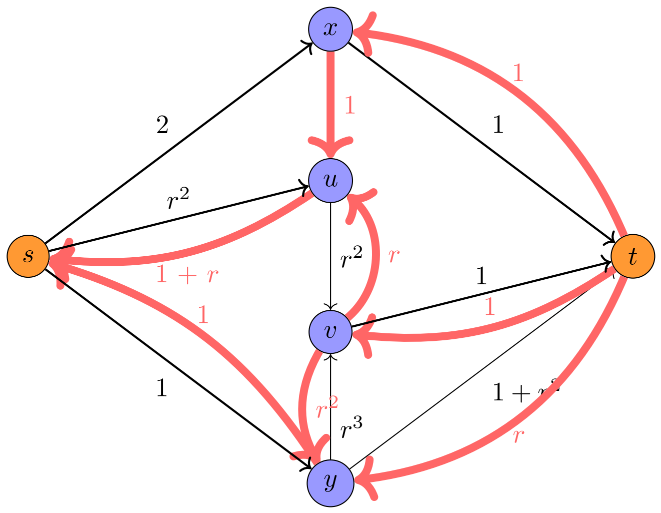
All the edges of the path $P2$ are present in the residual graph, allowing us to simulate it next.

4. After path $P2$:



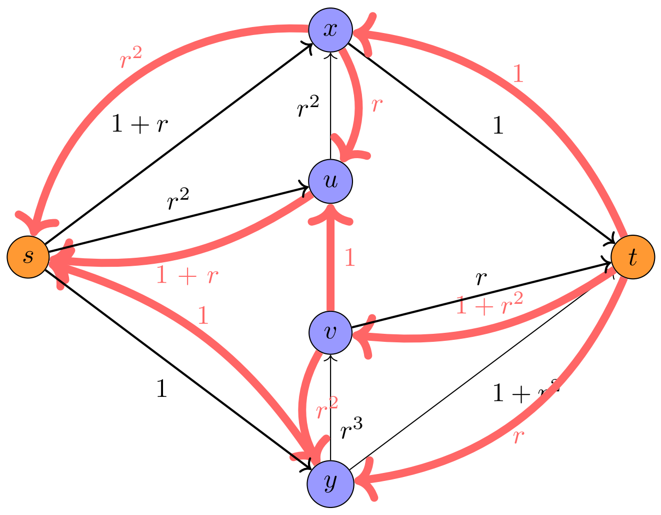
All the edges of the path $P1$ are present in the residual graph, allowing us to simulate it next.

5. After path $P1$:



All the edges of the path $P3$ are present in the residual graph, allowing us to simulate it next.

6. After path $P3$:



All the edges of the path $P3$ are present in the residual graph, allowing us to simulate it next.

From the graph, we can see that our induction hypothesis is satisfied for the base case. We assume that it is satisfied for the k^{th} application.

Induction Step

1. Application of path $P1$: The bottleneck capacity will be that of (y, v) since :

   - $$\begin{aligned}(s,y) &= 2-(r+r^2+\cdots+r^{2k-1}+r^{2k}) \\ &>= 2-\tfrac{r}{1-r} \\ &= 2-\tfrac{r}{r^2} \\ &= 2-\tfrac{1}{r} \\ &= 1-r = r^2 >= r^k\ \forall k>=2\end{aligned}$$
   - $(v,u) = 1-(u,v) = 1$
   - $(u,x) = r^{2k} > r^{2k+1}\ \forall\ k$
   - $$\begin{aligned}(x,t) &= 2-(r+r^2+\cdots+r^{2k-1}+r^{2k}) \\ &>= 2-\tfrac{r}{1-r} \\ &= 2-\tfrac{r}{r^2} \\ &= 2-\tfrac{1}{r} \\ &= 1-r = r^2 >= r^k\ \forall k>=2\end{aligned}$$

   Thus,
   $(s,y) = 2-(r+r^2+\cdots+r^{2k-1}+r^{2k})-r^{2k+1}$
   $(y,v) = 0$
   $(u,v) = r^{2k+1}$
   $(u,x) = r^{2k}-r^{2k+1} = r^{2k+2}$
   $(x,t) = 2-(r+r^2+\cdots+r^{2k-1}+r^{2k})-r^{2k+1}$

2. Application of path $P2$: The bottleneck capacity will be that of (u, v) since :

   - $$\begin{aligned}(s,u) &= 2-(r+r^3+\cdots+r^{2k-1}) \\ &>= 2-\tfrac{r}{1-r^2} \\ &= 2-\tfrac{r}{r} \\ &= 1 >= r^k\ \forall k>=0\end{aligned}$$
   - $(v,y) = r >= r^k\ \forall\ k>=1$
   - $$\begin{aligned}(y,t) &= 2-(r+r^3+\cdots+r^{2k-1}) \\ &>= 2-\tfrac{r}{1-r^2} \\ &= 2-\tfrac{r}{r} \\ &= 1 >= r^k\ \forall k>=0\end{aligned}$$

   Thus,
   $(s,u) = 2-(r+r^3+\cdots+r^{2k-1})-r^{2k+1}$
   $(u,v) = 0$
   $(y,v) = r^{2k+1}$
   $(y,t) = 2-(r+r^3+\cdots+r^{2k-1})-r^{2k+1}$

3. Application of path $P1$: The bottleneck capacity will be that of (u, x) since :

- $(s,y) = 2 - (r + r^2 + \cdots + r^{2k-1} + r^{2k}) - r^{2k+1}$
  $>= 2 - \frac{r}{1-r}$
  $= 2 - \frac{r}{r^2}$
  $= 2 - \frac{1}{r}$
  $= 1 - r = r^2 >= r^k \ \forall k >= 2$
- $(y,v) = r^{2k+1} > r^{2k+2}$
- $(v,u) = 1 - (u,v) = 1$
- $(x,t) = 2 - (r + r^2 + \cdots + r^{2k-1} + r^{2k}) - r^{2k+1}$
  $>= 2 - \frac{r}{1-r}$
  $= 2 - \frac{r}{r^2}$
  $= 2 - \frac{1}{r}$
  $= 1 - r = r^2 >= r^k \ \forall k >= 2$

Thus,
$(s,y) = 2 - (r + r^2 + \cdots + r^{2k-1} + r^{2k}) - r^{2k+1} - r^{2k+2}$
$(y,v) = r^{2k+1} - r^{2k+2} = r^{2k+3}$
$(u,v) = r^{2k+2}$
$(u,x) = 0$
$(x,t) = 2 - (r + r^2 + \cdots + r^{2k-1} + r^{2k}) - r^{2k+1} - r^{2k+2}$

4. Application of path $P3$: The bottleneck capacity will be that of (u,v) since :

- $(s,x) = 2 - (r^2 + r^4 + \cdots + r^{2k})$
  $>= 2 - \frac{r^2}{1-r^2}$
  $= 2 - \frac{r^2}{r}$
  $= 2 - r >= r^k \ \forall k >= 0$
- $(x,u) = 1 >= r^k \ \forall \ k$
- $(v,t) = 2 - (r^2 + r^4 + \cdots + r^{2k})$
  $>= 2 - \frac{r^2}{1-r^2}$
  $= 2 - \frac{r^2}{r}$
  $= 2 - r >= r^k \ \forall k >= 0$

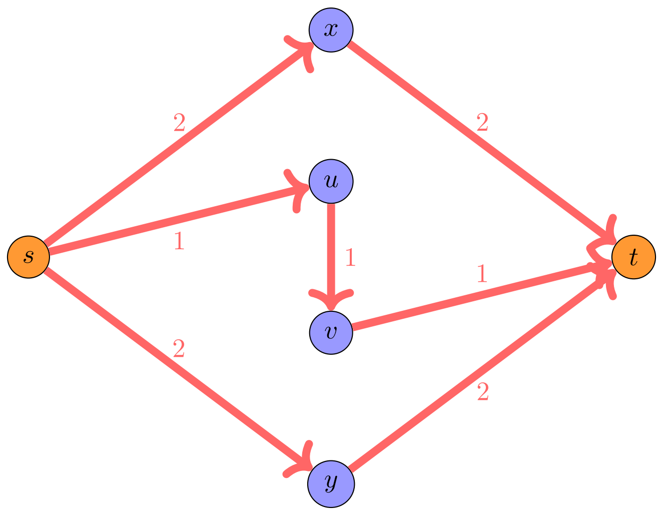
Thus,
$(s,x) = 2 - (r^2 + r^4 + \cdots + r^{2k}) - r^{2k+2}$
$(u,x) = r^{2k+2}$
$(u,v) = 0$
$(v,t) = 2 - (r^2 + r^4 + \cdots + r^{2k}) - r^{2k+2}$

Thus, we have shown that our induction hypothesis is true, since all the residual capacities calculated in the induction step agree with our initial claim. The algorithm never terminates since we are always able to select the given sequence. The flow added in the k^{th} sequence will be $2 * r^{2k-1} + 2 * r^{2k}$.

<u>Final Flow Calculation</u>
The flow upon selecting the given sequence will be :

$$\begin{aligned}
&1 + 2 * r + 2 * r^2 + 2 * r^3 + \ldots \\
&= 1 + 2 * (r + r^2 + r^3 + \ldots) \\
&= 1 + 2 * (\tfrac{r}{1-r}) \\
&= 1 + 2 * (\tfrac{r}{r^2}) \\
&= 1 + 2 * (\tfrac{1}{r}) \\
&= 1 + \sqrt{5} + 1 = 2 + \sqrt{5} < 5
\end{aligned}$$

However, consider the following flow diagram :



Clearly this is a legitimate flow in the graph G and has a value of 5. Thus, with the selected sequence of paths, the Ford-Fulkerson algorithm never terminates, and even the value it converges to is strictly less than the maximum flow.

Question 2.
Design a data structure to support the following three operations for a dynamic multi-set S of integers, which allows duplicate values:

- Insert(S, x): inserts x into S.
- Delete-Larger-Half(S): delete the largest $\left\lceil \frac{|S|}{2} \right\rceil$ elements from S.
- Report-Max(S): report the largest element from the set S.

Explain how to implement this data structure so that any sequence of m operations mentioned above run in $O(m)$ time. Your implementation should also include a way to output the elements of S in $O(|S|)$ time.

Answer.

Data Structure
Following is the data structure developed :

1. An dynamic table A storing the elements of multi-set S
2. A variable max storing the maximum of multi-set S

Following will be the operation methods :

1. Insert(S, x) :
   (a) Insertion method of dynamic table will be used for insertion in S:
      i. If A is not full, simple insertion takes place.
      ii. A will be expanded to twice the size when it becomes full, followed by a copy of all the elements in the table, followed by insertion.

   (b) max will be compared with x, and updated to x if $x > max$
2. Delete-Larger-Half(S) :
   (a) Median-of-medians algorithm will be used to find the element x such that there are exactly $\left\lceil \frac{|S|}{2} \right\rceil$ elements $>= x$
   (b) S is partitioned around x in linear time
   (c) The largest $\lceil \frac{|S|}{2} \rceil$ elements are then deleted, followed by the creation of a dynamic table A' such that $|A'| = \frac{|A|}{2}$, and elements are copied from A to A'
   (d) max is set to x
3. Report-Max(S): max is returned

Amortised Analysis
For the purpose of analysis, we consider the highest constant factor for any operation to be c, where $c = O(1)$. Amortised analysis is performed using the potential method. Following is the potential at any point of time :

$$\phi = c * (4 * |S| - |A|)$$

This is a valid potential function :

- $\phi(0) = 0$ since the multi-set is empty at the beginning, and $|S| = 0$, and the dynamic table is empty as well, i.e. $|A| = 0$
- $\phi(i) >= 0 \ \forall i$:
  1. Table expansion takes place only when the table is full, and the table is then expanded to half its current size. Thus, atleast half the table is full after each insertion.
  2. Since the table is reduced to half the size whenever S reduces to half its current value, the ratio of table occupied doesn't change on Delete-Larger-Half.

  Thus, at any point $2 * |S| >= |A|$. Hence, $c * (4 * |S| - |A|) >= 0$

Consider each operation :

1. Insert(S, x) :
   (a) Simple Insert :

i. Occurs when $|S| <= |A| - 1$
ii. Simple insert of element in A occurs, which has constant time complexity c
iii. It takes constant time c to update max
iv. $|S|$ increases by 1, while $|A|$ remains constant. Thus,

$$\Delta\phi = 4 * c$$

(b) Costly Insert :

i. Occurs when $|S| = |A|$
ii. Acquiring a new A' with $|A'| = 2 * |A|$ takes constant time a
iii. Copying $|S|$ elements from A' to A takes $b * |S|$ time
iv. Insertion of new element requires d time
v. Thus, total cost is $a + b * |S| + d <= c * |S|$
vi. It takes constant time c to update max
vii. $|S|$ increases to $|S| + 1$ while $|A|$ increases to $2 * |A|$. Thus,

$$\begin{aligned}\Delta\phi &= c * (4 * (|S| + 1) - 2 * |A|) - c * (4 * |S| - |A|)\\ &= -c * |A| + 4c\\ &= -c * |S| + 4c\end{aligned}$$

2. Delete-Larger-Half(S) :

(a) It takes linear time $e * |S|$ to find x, and a further time $f * |S|$ for partitioning.

(b) It takes $g * \lceil\frac{|S|}{2}\rceil$ time to delete the larger half

(c) Acquiring a new A' with $|A'| = \frac{|A|}{2}$ takes constant time a

(d) Copying $\lfloor\frac{|S|}{2}\rfloor$ elements takes $b * \lfloor\frac{|S|}{2}\rfloor$ time

(e) Thus, total cost is $e * |S| + f * |S| + g * \lceil\frac{|S|}{2}\rceil + a + b * \lfloor\frac{|S|}{2}\rfloor <= c * |S|$

(f) It takes constant time c to update max

(g) $|S|$ decreases by $\left\lceil\frac{|S|}{2}\right\rceil$ and $|A|$ decreases by $|A|/2$. Thus,

$$\begin{aligned}\Delta\phi &= -4 * c * \left\lceil\frac{|S|}{2}\right\rceil + c * \frac{|A|}{2}\\ &<= -4 * c * \frac{|S|}{2} + c * \frac{|A|}{2}\end{aligned}$$

Now, since $2 * |S| >= |A|$, $\frac{|A|}{2} <= |S|$. Hence,

$$\Delta\phi <= -c * |S|$$

3. Report-Max(S):

(a) It takes constant time c to report the maximum

(b) $|S|$ and $|A|$ don't change

(c) Thus, actual cost is c while $\Delta\phi = 0$

Operation	**Actual Cost**	**$\Delta(\phi)$**	**Amortized Cost**
Simple Insert(S, x)	$2 * c$	$4 * c$	$6 * c$
Costly Insert(S, x)	$c * \|S\| + c$	$-c * \|S\| + 4c$	$5 * c$
Delete-Larger-Half(S)	$c * \|S\| + c$	$-c * \|S\|$	c
Report-Max(S)	c	0	c

As can be seen from the table, the greatest amortised cost for all the operations is $6 * c = O(1)$.
Hence, amortised cost of m operations $= O(m) >=$ actual cost of m operations.
We can simply traverse the dynamic table A to output the elements of $|S|$ in $O(|S|)$ time.

Algorithm

Algorithm 1 Initialize Data Structure

```
1: function INITIALIZE
2:     Create dynamic table $A$ of size 0
3:     $max \leftarrow -\infty$
4:     return $(A, max)$
```

Algorithm 2 Insert Operation

```
function INSERT($S, x$)
    if $A$ is empty then
        Create new table $A$ of size 1
    else if $A$ is full then
        Create new table $A'$ of size $2 * |A|$
        Copy all elements from $A$ to $A'$
        $A \leftarrow A'$
    Append $x$ to $A$
    if $x > max$ then
        $max \leftarrow x$
```

Algorithm 3 Delete-Larger-Half Operation

```
function DELETELARGERHALF($S$)
    $n \leftarrow |S|$
    $k \leftarrow \lfloor \frac{n}{2} \rfloor$
    $x \leftarrow$ MEDIANOFMEDIANS($A, k$)
    $i \leftarrow 0$
    for $j \leftarrow 1$ to $n$ do
        if $A[j] \leq x$ then
            $i \leftarrow i + 1$
            Swap $A[i]$ with $A[j]$
    Delete largest $\lceil \frac{n}{2} \rceil$ elements from $A$
    if $A$ is empty then
        Delete $A$
        return
    Create new table $A'$ of size $\frac{|A|}{2}$
    Copy remaining elements from $A$ to $A'$
    $A \leftarrow A'$
    $max \leftarrow x$
```

Algorithm 4 Report-Max Operation

```
function REPORTMAX($S$)
    return $max$
```

Algorithm 5 Print Multi-set

```
function PRINTMULTISET($S$)
    for $i \leftarrow 1$ to $|A|$ do
        Print $A[i]$
```

Algorithm 6 Median of Medians Algorithm

```
function MedianOfMedians($A, k$)
    if $|A| \leq 5$ then
        Sort $A$
        return $A[k]$
    $groups \leftarrow \left\lceil \frac{|A|}{5} \right\rceil$
    Create array $M$ of size $groups$
    for $i \leftarrow 1$ to $groups$ do
        $start \leftarrow 5(i-1)+1$
        $end \leftarrow \min(5i, |A|)$
        $group \leftarrow A[start...end]$
        Sort $group$
        $M[i] \leftarrow group[\left\lceil \frac{|group|}{2} \right\rceil]$
    $pivot \leftarrow$ MedianOfMedians($M, \left\lceil \frac{|M|}{2} \right\rceil$)
    Create arrays $L$, $E$ for elements less than and equal to $pivot$
    for $x$ in $A$ do
        if $x < pivot$ then
            Append $x$ to $L$
        else if $x = pivot$ then
            Append $x$ to $E$
    if $k \leq |L|$ then
        return MedianOfMedians(L, k)
    else if $k \leq |L| + |E|$ then
        return $pivot$
    else
        Create array $G$ for elements greater than $pivot$
        for $x$ in $A$ do
            if $x > pivot$ then
                Append $x$ to $G$
        return MedianOfMedians(G, k$-|L| - |E|$)
```