**A+ Computer Science  
Queues M/C TEST**

**Directions ::** On your answer sheet, mark the letter of the best answer to each question.

1. **Which of the following adds and removes items according to FIFO rules?**

|  |  |
| --- | --- |
| a. | stack |
| b. | queue |
| c. | priority queue |
| d. | binary tree |
| e. | array |

2. **Which of the following adds and removes items according to comparisons made by the compareTo in the Objects being stored?**

|  |  |
| --- | --- |
| a. | stack |
| b. | queue |
| c. | priority queue |
| d. | binary tree |
| e. | array |

3. **What is output by the code below?**

Stack<Integer> a = new Stack<Integer>();

Queue<Integer> b;

b = new LinkedList<Integer>();

a.push(5);

a.push(7);

out.println(a); //line 1

|  |  |
| --- | --- |
| a. | [5] |
| b. | [7] |
| c. | [5,7] |
| d. | [7,5] |
| e. | [] |

4. **What is output by the code below?**

Stack<Integer> a = new Stack<Integer>();

Queue<Integer> b;

b = new LinkedList<Integer>();

a.push(5);

a.push(7);

b.add(a.pop());

b.add(a.pop());

out.println(b);

|  |  |
| --- | --- |
| a. | [5] |
| b. | [7] |
| c. | [5,7] |
| d. | [7,5] |
| e. | [] |

5. **What is output by the code below?**

Queue<Integer> c;

c = new LinkedList<Integer>();

c.add(9);

c.add(5);

c.add(3);

out.println(c.remove());

|  |  |
| --- | --- |
| a. | 1 |
| b. | 3 |
| c. | 5 |
| d. | 9 |
| e. | null |

6. **What is output by the code below?**

Queue<Integer> c;

c = new LinkedList<Integer>();

c.add(9);

c.add(5);

c.add(3);

c.add(17);

c.add(19);

out.println(c.peek());

|  |  |
| --- | --- |
| a. | 5 |
| b. | 19 |
| c. | 9 |
| d. | 17 |
| e. | null |

7. **What is output by the code below?**

Queue<Integer> c;

c = new LinkedList<Integer>();

c.add(9);

c.add(5);

c.add(3);

c.add(17);

c.add(19);

out.println(c.isEmpty());

|  |  |
| --- | --- |
| a. | 1 |
| b. | 0 |
| c. | true |
| d. | false |
| e. | null |

8. **What is output by the code below?**

Queue<Integer> c;

c = new LinkedList<Integer>();

c.add(9);

c.add(5);

c.add(3);

c.add(17);

c.remove(5);

out.println(c);

|  |  |
| --- | --- |
| a. | [9, 3, 17] |
| b. | [3, 17, 9] |
| c. | [17, 9, 3] |
| d. | [9, 17, 3] |
| e. | [3, 9 ,17] |

9. **What is output by the code below?**

Queue<Integer> d;

d = new LinkedList<Integer>();

d.add(11);

d.add(78);

d.add(23);

out.println(d.remove());

|  |  |
| --- | --- |
| a. | 78 |
| b. | 23 |
| c. | 14 |
| d. | 11 |
| e. | null |

10. **What is output by the code below?**

Queue<Integer> e;

e = new LinkedList<Integer>();

e.add(14);

e.add(53);

e.add(9);

e.add(5);

out.print(e.remove()+" ");

out.println(e.remove());

|  |  |
| --- | --- |
| a. | 14 9 |
| b. | 9 5 |
| c. | 53 9 |
| d. | 14 53 |
| e. | null |

11. **Which of the following statements would correctly fill <\*1>?**

public class StringQueue

{

private ArrayList<String> queue;

public StringQueue(){

queue=new ArrayList<String>();

}

public void add(String x){

queue.add(x);

}

public String remove(){

**<\*1>**

}

public String front(){

**<\*2>**

}

public String back(){

**<\*3>**

}

public boolean isEmpty(){

**<\*4>**

}

}

|  |  |
| --- | --- |
| a. | return queue.remove(0); |
| b. | return queue.remove(); |
| c. | return queue.get(0); |
| d. | return queue[0]; |
| e. | return queue.set(0,0); |

12. **Assuming <\*1> has been filled correctly, which of the following statements would correctly fill <\*2>?**

public class StringQueue

{

private ArrayList<String> queue;

public StringQueue(){

queue=new ArrayList<String>();

}

public void add(String x){

queue.add(x);

}

public String remove(){

**<\*1>**

}

public String front(){

**<\*2>**

}

public String back(){

**<\*3>**

}

public boolean isEmpty(){

**<\*4>**

}

}

|  |  |
| --- | --- |
| a. | return queue.get(queue.size()); |
| b. | return queue.get(queue.size()-1); |
| c. | return queue.get(0); |
| d. | return queue[0]; |
| e. | return queue[queue.size()-1]; |

13. **Assuming <\*1> and <\*2> have been filled correctly, which of the following statements would correctly fill <\*3>?**

public class StringQueue

{

private ArrayList<String> queue;

public StringQueue(){

queue=new ArrayList<String>();

}

public void add(String x){

queue.add(x);

}

public String remove(){

**<\*1>**

}

public String front(){

**<\*2>**

}

public String back(){

**<\*3>**

}

public boolean isEmpty(){

**<\*4>**

}

}

|  |  |
| --- | --- |
| a. | return queue.get(0); |
| b. | return queue.get(queue.size()-1); |
| c. | return queue.get(queue.size()); |
| d. | return queue[0]; |
| e. | return queue[queue.size()-1]; |

14. **Assuming <\*1> , <\*2> and <\*3> have been filled correctly, which of the following statements would correctly fill <\*4>?**

public class StringQueue

{

private ArrayList<String> queue;

public StringQueue(){

queue=new ArrayList<String>();

}

public void add(String x){

queue.add(x);

}

public String remove(){

**<\*1>**

}

public String front(){

**<\*2>**

}

public String back(){

**<\*3>**

}

public boolean isEmpty(){

**<\*4>**

}

}

|  |  |
| --- | --- |
| a. | return queue.length==0; |
| b. | return queue.size==0; |
| c. | return queue.size()==0; |
| d. | return queue.size; |
| e. | return queue.size()-1; |

15. **What is the output by the code below?**

String s = "rstuv";

Queue<Character> q;

q = new LinkedList<Character>();

for(char let : s.toCharArray())

{

q.add(let);

}

out.println(q.remove());

|  |  |
| --- | --- |
| a. | r |
| b. | s |
| c. | t |
| d. | u |
| e. | v |

16. **What is the output by the code below?**

String s = "rstuv";

Queue<Character> q;

q = new LinkedList<Character>();

for(char let : s.toCharArray())

{

q.add(let);

}

q.remove();

q.add('w');

q.add('x');

q.add('y');

q.add('z');

out.println(q.remove());

|  |  |
| --- | --- |
| a. | r |
| b. | s |
| c. | t |
| d. | u |
| e. | v |

17. **What is the output by the code below?**

String s = "rstuv";

Queue<Character> q;

q = new LinkedList<Character>();

for(char let : s.toCharArray())

{

q.add(let);

}

q.remove();

q.add('w');

q.add('x');

q.add('y');

q.add('z');

q.remove();

out.println(q.remove());

|  |  |
| --- | --- |
| a. | r |
| b. | s |
| c. | t |
| d. | u |
| e. | v |

18. **After this code segment is executed, how many elements remain in q?**

String s = "rstuv";

Queue<Character> q;

q = new LinkedList<Character>();

for(char let : s.toCharArray())

{

q.add(let);

}

q.remove();

q.add('w');

q.add('x');

q.add('y');

q.add('z');

q.remove();

q.remove();

|  |  |
| --- | --- |
| a. | 6 |
| b. | 4 |
| c. | 5 |
| d. | 0 |
| e. | 2 |

19. **What is output by the code below?**

String word = "guxys";

PriorityQueue<Character> theQ;

theQ = new PriorityQueue<Character>();

for(char ch : word.toCharArray())

{

theQ.add(ch);

}

out.println(theQ.remove());

|  |  |
| --- | --- |
| a. | r |
| b. | s |
| c. | t |
| d. | u |
| e. | g |

20. **What is output by the code below?**

String word = "guxys";

PriorityQueue<Character> theQ;

theQ = new PriorityQueue<Character>();

for(char ch : word.toCharArray())

{

theQ.add(ch);

}

theQ.remove();

theQ.remove();

theQ.remove();

theQ.remove();

theQ.add('r');

theQ.add('t');

theQ.add('z');

theQ.add('y');

out.println(theQ.remove());

|  |  |
| --- | --- |
| a. | r |
| b. | s |
| c. | t |
| d. | u |
| e. | g |

21. **What is output by the code below?**

String word = "guxys";

PriorityQueue<Character> theQ;

theQ = new PriorityQueue<Character>();

for(char ch : word.toCharArray())

{

theQ.add(ch);

}

theQ.remove();

theQ.remove();

theQ.remove();

theQ.remove();

theQ.add('r');

theQ.add('t');

theQ.add('z');

theQ.add('y');

theQ.remove();

out.println(theQ.remove()); //line 3

|  |  |
| --- | --- |
| a. | r |
| b. | s |
| c. | t |
| d. | u |
| e. | g |

22. **What is output by the code below?**

String word = "guxys";

PriorityQueue<Character> theQ;

theQ = new PriorityQueue<Character>();

for(char ch : word.toCharArray())

{

theQ.add(ch);

}

theQ.remove();

theQ.remove();

theQ.remove();

theQ.remove();

theQ.add('r');

theQ.add('t');

theQ.add('z');

theQ.add('y');

theQ.remove();

theQ.remove();

out.println(theQ.size());

|  |  |
| --- | --- |
| a. | 6 |
| b. | 4 |
| c. | 5 |
| d. | 3 |
| e. | 2 |

23. **What is output by the code below?**

PriorityQueue<Double> priorQ;

priorQ = new PriorityQueue<Double>();

priorQ.add(8.5);

priorQ.add(4.5);

out.println(priorQ.remove());

|  |  |
| --- | --- |
| a. | 4.5 |
| b. | 8.5 |
| c. | 5.5 |
| d. | 6.7 |
| e. | 9.5 |

24. **What is output by the code below?**

PriorityQueue<Double> priorQ;

priorQ = new PriorityQueue<Double>();

priorQ.add(8.5);

priorQ.add(4.5);

priorQ.remove();

priorQ.add(5.5);

priorQ.add(9.5);

out.println(priorQ.remove());

|  |  |
| --- | --- |
| a. | 4.5 |
| b. | 8.5 |
| c. | 5.5 |
| d. | 6.7 |
| e. | 9.5 |

25. **What is output by the code below?**

PriorityQueue<Double> priorQ;

priorQ = new PriorityQueue<Double>();

priorQ.add(8.5);

priorQ.add(4.5);

priorQ.remove();

priorQ.add(5.5);

priorQ.add(9.5);

priorQ.remove();

priorQ.add(6.7);

priorQ.add(11.5);

out.println(priorQ.remove());

|  |  |
| --- | --- |
| a. | 4.5 |
| b. | 8.5 |
| c. | 5.5 |
| d. | 6.7 |
| e. | 9.5 |

26. **What is output by the code below?**

PriorityQueue<String> pQ;

pQ = new PriorityQueue<String>();

pQ.add("d");

pQ.add("a");

out.println(pQ.remove());

|  |  |
| --- | --- |
| a. | d |
| b. | a |
| c. | w |
| d. | r |
| e. | z |

27. **What is output by the code below?**

PriorityQueue<String> pQ;

pQ = new PriorityQueue<String>();

pQ.add("d");

pQ.add("a");

pQ.remove();

pQ.add("w");

pQ.add("z");

out.println(pQ.remove());

|  |  |
| --- | --- |
| a. | d |
| b. | a |
| c. | w |
| d. | r |
| e. | z |

28. **What is output by the code below?**

PriorityQueue<String> pQ;

pQ = new PriorityQueue<String>();

pQ.add("d");

pQ.add("a");

pQ.remove();

pQ.add("w");

pQ.add("z");

pQ.remove();

pQ.add("r");

out.println(pQ.remove());

|  |  |
| --- | --- |
| a. | d |
| b. | a |
| c. | w |
| d. | r |
| e. | z |

29. **What is output by the code below?**

PriorityQueue<String> pQ;

pQ = new PriorityQueue<String>();

pQ.add("d");

pQ.add("a");

pQ.remove();

pQ.add("z");

pQ.add("w");

pQ.add("b");

pQ.remove();

out.println(pQ);

|  |  |
| --- | --- |
| a. | [d, w, z] |
| b. | [d, z, w] |
| c. | [z, d, w] |
| d. | [b, w, d] |
| e. | [b, d, w] |

30. **What is output by the code below?**

PriorityQueue<Integer> pQ;

pQ = new PriorityQueue<Integer>();

for (int i = 5; i > 0; i--)

pQ.add(i);

out.println(pQ);

|  |  |
| --- | --- |
| a. | [1, 2, 3, 4, 5] |
| b. | [1, 3, 2, 5, 4] |
| c. | [1, 3, 4, 5, 2] |
| d. | [1, 2, 5, 3, 4] |
| e. | [1, 2, 4, 5, 3] |

31. **What is output by the code below?**

PriorityQueue<Integer> pQ;

pQ = new PriorityQueue<Integer>();

for (int i = 1; i <= 5; i++)

pQ.add(i);

out.println(pQ);

|  |  |
| --- | --- |
| a. | [1, 2, 3, 4, 5] |
| b. | [1, 3, 2, 5, 4] |
| c. | [1, 3, 4, 5, 2] |
| d. | [1, 2, 5, 3, 4] |
| e. | [1, 2, 4, 5, 3] |

32. **What is the running time of the enqueue method in this queue implementation?**

class AplusQ {

Object[] stuff;

int head, tail;

public AplusQ() {

this(10);

}

public AplusQ(int l) {

stuff = new Object[l];

head = 0;

tail = 0;

}

public boolean enqueue(Object x) {

if ( **<\*1>**  ) // overflow

return false;

stuff[tail++ % stuff.length] = x;

return true;

}

public Object dequeue() {

if ( **<\*2>**  ) // empty queue

return null;

return stuff[head++ % stuff.length];

}

}

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N^2) |
| d. | O(2^N) |
| e. | O(N!) |

33. **What is the running time of the dequeue method in this queue implementation?**

class AplusQ {

Object[] stuff;

int head, tail;

public AplusQ() {

this(10);

}

public AplusQ(int l) {

stuff = new Object[l];

head = 0;

tail = 0;

}

public boolean enqueue(Object x) {

if ( **<\*1>**  ) // overflow

return false;

stuff[tail++ % stuff.length] = x;

return true;

}

public Object dequeue() {

if ( **<\*2>**  ) // empty queue

return null;

return stuff[head++ % stuff.length];

}

}

|  |  |
| --- | --- |
| a. | O(1) |
| b. | O(N) |
| c. | O(N^2) |
| d. | O(2^N) |
| e. | O(N!) |

34. **Which of the following statements would correctly fill <\*1>?**

class AplusQ {

Object[] stuff;

int head, tail;

public AplusQ() {

this(10);

}

public AplusQ(int l) {

stuff = new Object[l];

head = 0;

tail = 0;

}

public boolean enqueue(Object x) {

if ( **<\*1>**  ) // overflow

return false;

stuff[tail++ % stuff.length] = x;

return true;

}

public Object dequeue() {

if ( **<\*2>**  ) // empty queue

return null;

return stuff[head++ % stuff.length];

}

}

|  |  |
| --- | --- |
| a. | tail - stuff.length <= head |
| b. | tail - stuff.length < head |
| c. | tail - stuff.length >= head |
| d. | tail - stuff.length > head |
| e. | tail >= head |

35. **Assuming <\*1> has been filled correctly, which of the following statements would correctly fill <\*2>?**

class AplusQ {

Object[] stuff;

int head, tail;

public AplusQ() {

this(10);

}

public AplusQ(int l) {

stuff = new Object[l];

head = 0;

tail = 0;

}

public boolean enqueue(Object x) {

if ( **<\*1>**  ) // overflow

return false;

stuff[tail++ % stuff.length] = x;

return true;

}

public Object dequeue() {

if ( **<\*2>**  ) // empty queue

return null;

return stuff[head++ % stuff.length];

}

}

|  |  |
| --- | --- |
| a. | tail - stuff.length < head |
| b. | tail > head |
| c. | head > tail |
| d. | head >= tail |
| e. | head > stuff.length |

36. **What is output by the code below?**

Queue<Integer> c;

c = new LinkedBlockingQueue<Integer>(3);

for (int j = 0; j < 5; j++)

c.offer(j);

out.println(c);

|  |  |
| --- | --- |
| a. | [0, 1, 2] |
| b. | [2, 3, 4] |
| c. | [1, 2, 3] |
| d. | [0, 1, 2, 3, 4] |
| e. | an exception is thrown |

37. What is the output by the code below?

public static int fun (Map<Integer, int[]> am, int start, int end)

{

Queue<int[]> aplus = new LinkedList<int[]>();

Set<Integer> comp = new TreeSet<Integer>();

aplus.add(new int[]{start, 0});

while (!aplus.isEmpty())

{

int[] tmp = aplus.remove();

int[] con = am.get(tmp[0]);

if (tmp[0] == end)

return tmp[1];

if (!comp.add(tmp[0]) || con == null)

continue;

for (int item : con) {

aplus.add(new int[]{item , tmp[1] + 1});

}

}

return -1;

}

**// client code**

Map<Integer, int[]> am;

am = new TreeMap<Integer, int[]>();

am.put(1, new int[]{2, 4});

am.put(2, new int[]{3, 5});

am.put(3, new int[]{6, 1});

am.put(4, new int[]{6});

out.println(fun(am, 3, 7));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

38. What is the output by the code below?

public static int fun (Map<Integer, int[]> am, int start, int end)

{

Queue<int[]> aplus = new LinkedList<int[]>();

Set<Integer> comp = new TreeSet<Integer>();

aplus.add(new int[]{start, 0});

while (!aplus.isEmpty())

{

int[] tmp = aplus.remove();

int[] con = am.get(tmp[0]);

if (tmp[0] == end)

return tmp[1];

if (!comp.add(tmp[0]) || con == null)

continue;

for (int item : con) {

aplus.add(new int[]{item , tmp[1] + 1});

}

}

return -1;

}

**// client code**

Map<Integer, int[]> am;

am = new TreeMap<Integer, int[]>();

am.put(1, new int[]{2, 4});

am.put(2, new int[]{3, 5});

am.put(3, new int[]{6, 1});

am.put(4, new int[]{6});

out.println(fun(am, fun(am, 1, 6), 5));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

39. What is the output by the code below?

public static int fun (Map<Integer, int[]> am, int start, int end)

{

Queue<int[]> aplus = new LinkedList<int[]>();

Set<Integer> comp = new TreeSet<Integer>();

aplus.add(new int[]{start, 0});

while (!aplus.isEmpty())

{

int[] tmp = aplus.remove();

int[] con = am.get(tmp[0]);

if (tmp[0] == end)

return tmp[1];

if (!comp.add(tmp[0]) || con == null)

continue;

for (int item : con) {

aplus.add(new int[]{item , tmp[1] + 1});

}

}

return -1;

}

**// client code**

Map<Integer, int[]> am;

am = new TreeMap<Integer, int[]>();

am.put(1, new int[]{2, 4});

am.put(2, new int[]{3, 5});

am.put(3, new int[]{6, 1});

am.put(4, new int[]{6});

out.println(fun(am, fun(am, 1, 5), 1));

|  |  |
| --- | --- |
| a. | 1 |
| b. | 2 |
| c. | 3 |
| d. | 4 |
| e. | -1 |

40. What does method fun do?

public static int fun (Map<Integer, int[]> am, int start, int end)

{

Queue<int[]> aplus = new LinkedList<int[]>();

Set<Integer> comp = new TreeSet<Integer>();

aplus.add(new int[]{start, 0});

while (!aplus.isEmpty())

{

int[] tmp = aplus.remove();

int[] con = am.get(tmp[0]);

if (tmp[0] == end)

return tmp[1];

if (!comp.add(tmp[0]) || con == null)

continue;

for (int item : con) {

aplus.add(new int[]{item , tmp[1] + 1});

}

}

return -1;

}

|  |  |
| --- | --- |
| a. | Build a minimum spanning tree |
| b. | Traverses all the nodes |
| c. | Finds the number of paths between two nodes |
| d. | Finds the shortest path between two nodes |
| e. | Deletes the connections between nodes |