

*Puede entregar las respuestas a las partes que considere oportuno.*

1. 16 cuestiones tipo test para el parcial 1
2. 2 ejercicios breves de respuesta abierta para la práctica 2
3. 16 cuestiones tipo test para el parcial 2

*Cada cuestión tipo test plantea 4 alternativas y tiene una única respuesta correcta. Cada respuesta correcta aporta 10/16 puntos, y cada error descuenta 10/48 puntos. Debe contestar las cuestiones tipo test en la hoja de respuestas.*

### PARCIAL 1

**1** *Los clusters altamente disponibles son un ejemplo de área de aplicación de los sistemas distribuidos en la que...:*

- a** Se utiliza replicación para mejorar el rendimiento y la disponibilidad del servicio.
- b** Se distribuye la responsabilidad de servicio entre los nodos 'clientes', convirtiéndolo en cooperativo y fácilmente escalable.
- c** Todas las demás opciones son ciertas.
- d** Su modelo de servicio automatiza el despliegue de las aplicaciones distribuidas.

**2** *En la computación en la nube, el objetivo principal del modelo de servicio SaaS es:*

- a** Bajo un modelo de pago por uso, facilitar la infraestructura necesaria y administrarla debidamente.
- b** Automatizar las tareas de despliegue de servicios a sus clientes, sin que estos deban preocuparse tampoco por la infraestructura subyacente.
- c** Ofrecer un conjunto de aplicaciones distribuidas ya desarrolladas, para que los clientes las adquieran y las desplieguen allí donde prefieran.
- d** Ofrecer servicios de cómputo remotos a sus clientes, para que estos últimos no deban preocuparse del despliegue de las aplicaciones ni la administración de los equipos.

**3** *La Wikipedia es un buen caso de estudio dentro del Tema 1 porque:*

- a** Es un servicio distribuido que ilustra todas las etapas de la evolución de los servicios; p. ej., utilizó 'mainframes' en su primera etapa y es un SaaS en su etapa actual.
- b** Ha sido programada en Node.js (JavaScript) al igual que los proyectos a desarrollar en las prácticas de TSR.
- c** Todas las demás opciones son ciertas.
- d** Utiliza múltiples mecanismos para que un servicio distribuido sea escalable y proporciona cierta documentación sobre ellos.

Considérese el siguiente conjunto de programas JavaScript:

```
// Program: ex1.js
const ev = require('events')
const emitter = new ev.EventEmitter()
const el = "print"
emitter.on(el, function(num) {
  return () =>
    console.log("Event " + el + ": " + ++num)
})(0)
emitter.emit(el)
```

```
// Program: ex2.js
function f(x){
  return function (y) {
    let z = x + y
    return z
  }
}
const f2 = f(10)
```

4 En la ejecución de 'ex1.js', si no se considera el hilo inicial de ejecución, ¿cuántas veces habrá algún 'listener' de el en la cola de eventos?

- a Ninguna de las demás opciones es cierta.
- b Un número indefinido de veces, pues el evento el se genera repetidamente.
- c Una vez.
- d Ninguna.

5 ¿Qué ámbito tiene la variable z en el programa 'ex2.js'?

- a Ninguno, pues su declaración es errónea.
- b Global.
- c Local a la función anónima contenida en la función f.
- d Local a la función f.

6 ¿Se define alguna clausura en los programas 'ex1.js' y 'ex2.js'?

- a Ninguna.
- b Sí, pero solo en 'ex2.js'.
- c Sí, una en cada programa.
- d Sí, pero solo en 'ex1.js'.

7 El siguiente fragmento de código escribe cierto contenido en 3 ficheros y después los lee utilizando diferentes versiones del API de leer ficheros.

Suponiendo que ejecutamos el programa y que la escritura de los 3 ficheros se completa sin errores, indique la afirmación correcta:

```
const fs=require('fs')
const fsp=fs.promises;

// Escribe el contenido "fb", "fc" y "fd" en
// ficheros "fb", "fc" y "fd" respectivamente
let write = (x) => fs.writeFileSync (x, x)
write ("fb"); write ("fc"); write ("fd")

// Leemos de los ficheros
fsp.readFile ("fb").then ( data =>
  console.log (data + ""))
fs.readFile ("fc", (err,data) =>
  console.log (data + ""))
console.log(""+fs.readFileSync ("fd"))
```

- a Es posible que al ejecutar el programa, veamos el contenido de los ficheros en este orden: fb, fc, fd
- b Al ejecutar el programa únicamente veremos el contenido de los ficheros 'fc' y 'fd'. El contenido de 'fb' no lo veremos.
- c Es posible que al ejecutar el programa, veamos el contenido de los ficheros en este orden: fd, fc, fb
- d Al ejecutar el programa únicamente veremos el contenido de los ficheros 'fb' y 'fc'. El contenido de 'fd' no lo veremos.

- 8** En el tema 3 se afirma que 'El contenido de los mensajes resulta transparente para ØMQ', pero...
- a** No es posible enviar informaciones de varios tipos en el mismo mensaje
  - b** Es posible enviar informaciones de varios tipos incluso en el mismo segmento
  - c** La responsabilidad de interpretar los tipos y contenido de un mensaje corre a cargo de los participantes
  - d** Solo se pueden emplear varios segmentos en un mensaje si se necesita incluir datos de varios tipos
- 9** En ØMQ, ¿cómo podrían dos procesos publicadores diferentes difundir información a varios suscriptores si estos últimos solo utilizan un socket SUB cada uno?
- a** No es posible hacerlo porque habría dos operaciones bind para el mismo punto de conexión al que los suscriptores harían connect.
  - b** Es una facilidad aplicable exclusivamente al caso en que un único suscriptor hace bind y todos los publicadores connect (a la url del suscriptor).
  - c** Cada suscriptor debería realizar varios connect: uno por cada publicador.
  - d** Se puede conseguir si la url empleada por los suscriptores incluye una expresión regular (p.ej un asterisco) para poder referenciar todos los publicadores implicados.

**10** En ØMQ:

- a** Los sockets REQ solo tienen cola de envío de mensajes.
- b** Los sockets REP solo tienen cola de recepción de mensajes.
- c** Tanto los sockets REQ como los REP tienen colas de envío y de recepción de mensajes.
- d** De las otras tres respuestas, solo dos son correctas.

Considere los dos programas siguientes:

```
// File: sender.js
const zmq = require('zeromq')
const push = zmq.socket('push')
const NUM_MSGS = 10
const DELAY = 1000
push.connect('tcp://127.0.0.1:8000')
let count = 0
function sender() {
  push.send("Message number " + ++count +
    " from sender " + process.pid)
  if (count < NUM_MSGS)
    setTimeout( sender, DELAY )
  else push.close()
}
sender()
```

```
// File: receiver.js
const zmq = require('zeromq')
const pull = zmq.socket('pull')
pull.bind('tcp://127.0.0.1:8000', (err) => {
  if (err) {
    console.log("Error on bind()")
    process.exit(1)
  }
})
pull.on('message', (msg) => {
  console.log(msg+" ")
})
```

- 11** Considere los programas sender.js y receiver.js originales. Suponga que el puerto 8000 está libre.

Lanzamos un proceso emisor que ejecuta el programa sender.js y al cabo de cinco segundos se inicia otro proceso receptor (receiver.js) en ese mismo ordenador.

¿Qué sucede con los mensajes enviados en ese canal de comunicación?

- a** Se pierden todos, pues el emisor no llega a conectar nunca con el receptor.
- b** Se entregan todos al receptor y este muestra el contenido de los diez mensajes en pantalla a medida que los recibe.
- c** Se pierden los cinco o seis primeros mensajes, dependiendo del instante concreto en que pueda establecerse la conexión entre ambos procesos.
- d** Ninguna de las demás opciones es correcta.

**12** En los programas *sender.js* y *receiver.js*.

¿Cuántos mensajes llegaría a mostrar el receptor por pantalla si sustituyéramos el socket *PUSH* del emisor por un *REQ* y el *PULL* del receptor por un *REP*, sin hacer ningún otro cambio en ambos programas?

Suponga que los nuevos emisor y receptor son iniciados a la vez en una misma máquina y que el puerto 8000 estaba libre antes de iniciar ambos procesos.

- a** Ninguno, pues el emisor no llega a conectar nunca con el receptor.
- b** El receptor muestra el contenido de los diez mensajes en pantalla, pues llega a recibir cada uno a su debido tiempo.
- c** Seguro que muestra desde el segundo al décimo mensaje y puede que también el primero, pero este depende del instante en que lleguen a conectarse los procesos.
- d** Ninguna de las demás opciones es correcta.

**13** Considere el siguiente programa JavaScript:

```
for (var i=0; i<5; i++) {  
  console.log ("i: " + i)  
}  
console.log ("fin --> i=" + i)
```

¿Qué cambios habría en la ejecución del programa si sustituimos la palabra reservada *'var'* por *'let'* en su primera línea?

- a** El programa se comportaría de igual manera, pues *'i'* sigue siendo una variable global.
- b** El programa generaría un error en su última línea.
- c** El programa generaría un error en su primera línea, pues *'let'* no puede utilizarse en los bucles *'for'*.
- d** Ninguna de las demás opciones es correcta.

**14** ¿Qué visualizaría en pantalla la ejecución del siguiente programa JavaScript?

```
function f1 (a,b,c) {  
  console.log (arguments.length +  
    " arguments")  
  return a+b+c;  
}  
console.log("result: " + f1(1,"a",[3,0],4,5))
```

- a** Un error.
- b** 5 arguments  
result: 1a3,0
- c** 6 arguments  
result: 1a3
- d** Ninguna de las demás opciones es correcta.

**15** Considere el siguiente programa JavaScript:

```
const fs=require("fs")  
console.log("Call to readFile")  
fs.readFile("a.txt",(e,d)=> {  
  if (e) console.error("readFile error")  
  else console.log(d+'')  
})  
console.log("End of readFile")  
  
console.log("Call to readFileSync")  
try {  
  console.log( fs.readFileSync("a.txt")+ "")  
} catch (e) {}  
console.log("End of readFileSync")
```

¿Cuál será la última cadena que este programa mostrará, si el fichero que se intenta leer no existe?

- a** 'End of readFile'
- b** 'readFile error'
- c** 'End of readFileSync'
- d** Ninguna de las demás opciones es correcta.

- 16** *Este es el programa inicial a completar en el ejemplo 'emisor3.js' de la Práctica 1:*

```
...
const e1='e1', e2='e2'
let inc=0, t
function rand() { // devolverá valor aleatorio
  // en rango [2000,5000) (ms)
  ... // Math.floor(x) parte entera del valor x
  ... // Math.random() valor aleatorio rango [0,1)
}
function handler (e,n) { // e es el evento, n
  // el valor asociado
  return (inc) => {...} // el oyente recibe un valor
}
emitter.on(e1, handler(e1,0))
emitter.on(e2, handler(e2,''))
function etapa() {
  ...
}
setTimeout(etapa,t=rand())
```

*En ese ejercicio debía programarse una serie indefinida de etapas, con duración aleatoria entre 2 y 5 segundos. Si en lugar de tener infinitas etapas, se solicitara generar 10 etapas y cada etapa no necesitara mostrar su duración en pantalla,*

*¿sería válido sustituir la última línea del programa facilitado por este bloque?*

```
t = 0
for (let i=0; i<10; i++) {
  t = t + rand()
  setTimeout (etapa, t)
}
```

- a** Sí.
- b** No, pues el primer argumento de `setTimeout`, tanto en el original como en este nuevo código, debería ser `etapa()` en lugar de `etapa`.
- c** No, pues no se consigue que la duración de cada etapa sea distinta y aleatoria.
- d** No, pues la variable 'i' debería haberse declarado con 'var' en lugar de 'let'.

## PARCIAL 2

- 17** *Tenemos un programa cliente.js que utiliza un socket s1 de tipo DEALER para interactuar, **utilizando mensajes con un único segmento**, (p.ej., `s1.send('request')`) con otro programa servidor.js que utiliza un socket s2 de tipo ROUTER. La comunicación se realiza sin problemas si ejecutamos un proceso de cada tipo.*

*Tiempo después se decide realizar un único cambio en esos programas, concretamente en cliente.js: sustituir `s1=zmq.socket('DEALER')` por `s1=zmq.socket('REQ')`.*

*Al comprobar el funcionamiento del nuevo cliente.js, se observa que servidor.js solo recibe mensajes aparentemente vacíos.*

*¿A qué puede deberse eso?*

- a** Al crear un canal REQ-ROUTER, el socket REQ debería tener unas colas de envío de gran capacidad. De no ser así, el contenido del mensaje se pierde al enviarlo.
- b** El socket ROUTER, si se conecta con un socket REQ, descarta automáticamente el primer segmento de cada mensaje recibido.
- c** El socket REQ añade un segmento inicial vacío al enviar los mensajes.
- d** Ninguna de las demás afirmaciones es cierta.

- 18** *Se desea desarrollar y desplegar una aplicación distribuida en la que habrá múltiples instancias de un componente A y al menos una instancia de un componente B.*

*A iniciará la comunicación, enviará peticiones, y esperará respuesta de B. B jamás iniciará la comunicación.*

*Se pretende que la comunicación entre A y B sea bidireccional y asíncrona, utilizando un único socket ØMQ en cada componente. ¿Qué sockets podrían usar A y B?*

- a** ROUTER en A y ROUTER en B.
- b** DEALER en A y ROUTER en B.
- c** DEALER en A y DEALER en B.
- d** Todas las demás opciones son válidas.

- 19** Considere una imagen 'a' en la que no existe ningún soporte para Node.js y este Dockerfile:

```
FROM a
COPY myPrg.js /
CMD node /myPrg
```

*Mediante la orden apropiada, se ha generado una imagen docker llamada new-a utilizando ese Dockerfile.*

*Seleccione la afirmación cierta de entre las siguientes:*

- a** Al generar la imagen new-a se ha instalado en ella el intérprete node.
- b** Para que no haya errores en esa generación de la imagen new-a, debe existir un fichero myProg.js en el mismo directorio donde esté el Dockerfile.
- c** Si no ha habido errores al generar la imagen new-a, la ejecución de la orden docker run new-a en ese mismo anfitrión seguro que no provocará ningún error.
- d** Todas las demás afirmaciones son ciertas.

- 20** Considere este Dockerfile:

```
FROM a
COPY myPrg.js /
CMD node /myPrg
```

*Para generar una imagen new-a con este Dockerfile se debería utilizar esta orden en su mismo directorio:*

- a** docker run Dockerfile
- b** docker commit new-a
- c** docker build -t new-a .
- d** Todas las demás afirmaciones son ciertas.

- 21** Existen dos alternativas para crear imágenes Docker: crearlas de manera interactiva en un contenedor que después guardaremos como imagen, o generarlas mediante un fichero Dockerfile.

*¿Por qué suele preferirse utilizar ficheros Dockerfile?*

- a** Porque permiten utilizar variables de entorno cuyo valor podrá proporcionarse al iniciar los contenedores, simplificando así la resolución de dependencias.
- b** Hay un número mayor de imágenes a tomar como base en los Dockerfile que generando la imagen de manera interactiva.
- c** Las imágenes generadas ocupan menos espacio y requieren menos recursos que las creadas de manera interactiva.
- d** Las imágenes generadas de manera interactiva no podrían utilizarse como imagen base para generar otras imágenes mediante ficheros Dockerfile.

- 22** ¿Por qué, entre otras razones, resulta más sencillo el despliegue de una aplicación distribuida si se utilizan contenedores que si se realiza la instalación y configuración de los diferentes programas que componen la aplicación manualmente?

- a** Porque las dependencias relacionadas con las bibliotecas necesarias y la configuración de cada programa se han resuelto en gran medida al crear sus imágenes.
- b** Porque los contenedores automatizan la intercomunicación de componentes, sin importar qué plan de despliegue se utilice.
- c** Porque los contenedores, por su configuración más precisa, siempre proporcionarán mayores garantías de seguridad.
- d** Todas las demás afirmaciones son falsas.

**23** Considere este fichero `docker-compose.yml`:

```
version: '2'
services:
  one:
    image: zzz
    links:
      - two
    environment:
      - SERVER_IP=two
      - SERVER_PORT=80
  two:
    image: yyy
    expose:
      - "80"
```

*Seleccione la afirmación verdadera, sobre la funcionalidad de la orden 'docker-compose up', en caso de que se utilice allí donde esté ese fichero docker-compose.yml:*

- a** Entre otras cosas, comprueba si en la imagen 'yyy' se realiza un bind() o listen() sobre el puerto 80. Solo inicia instancias de 'two' si eso se cumple.
- b** Entre otras cosas, comprueba si en la imagen 'zzz' se realiza un connect() sobre el puerto SERVER\_PORT de una máquina con dirección IP SERVER\_IP. Solo inicia instancias de 'one' si eso se cumple.
- c** Todas las demás afirmaciones son ciertas.
- d** Entre otras cosas, asignar la dirección IP de la instancia del servicio 'two' a la variable SERVER\_IP de cada instancia del servicio 'one'.

**24** Considere este fichero `docker-compose.yml`:

```
version: '2'
services:
  one:
    image: zzz
    links:
      - two
    environment:
      - SERVER_IP=two
      - SERVER_PORT=8080
  two:
    image: yyy
    expose:
      - "8080"
      - "8443"
```

*Seleccione la afirmación verdadera si no hubiera ninguna imagen 'zzz', pero sí una imagen 'yyy', en el ordenador anfitrión cuando se utilice la orden 'docker-compose up' allí donde reside el fichero:*

- a** Se buscará un fichero Dockerfile en el subdirectorio zzz y con él se generará la imagen zzz para iniciar una instancia de one.
- b** Se comprobará que la imagen zzz no existe localmente y se descargará del repositorio global, si allí existiera, para iniciar una instancia de one.
- c** Se comprobará que la imagen zzz no existe localmente, pero la cláusula links: indica que de ser así podremos usar la imagen yyy en su lugar.
- d** Ninguna de las demás afirmaciones es verdadera.

**25** Seleccione la afirmación correcta sobre los modelos de consistencia:

- a** El modelo FIFO es más fuerte que el modelo caché.
- b** El modelo caché es más fuerte que el modelo FIFO.
- c** El modelo causal es más fuerte que el modelo secuencial.
- d** Ninguna de las demás afirmaciones es verdadera.

- 26** *La replicación activa no suele utilizarse en las bases de datos relacionales porque:*
- a** Cada consulta suele requerir un largo intervalo de procesamiento y es común realizar muchas más consultas que modificaciones.
  - b** Los UPDATEs suelen ser más frecuentes que los SELECTs y pueden modificar una gran cantidad de estado.
  - c** No hay ninguna garantía de que la red de interconexión de las réplicas sea extremadamente rápida y una red rápida es imprescindible en el modelo activo.
  - d** Ninguna de las demás afirmaciones es cierta, pues todas las BBDD relacionales utilizan replicación activa, como ya vimos al analizar la Wikipedia.
- 27** *¿Qué afirmación es cierta para un servicio distribuido escalable?*
- a** El servicio podrá tolerar las situaciones de particionado de la red, mantener su disponibilidad y respetar una consistencia relajada simultáneamente.
  - b** El servicio no podrá respetar la consistencia secuencial.
  - c** El servicio, simultáneamente, tolera las particiones de la red, ofrece disponibilidad y respeta la consistencia estricta entre todos los subgrupos que puedan formarse.
  - d** Para que el servicio respete la consistencia FIFO y esté disponible deberá evitarse que haya particiones en la red.
- 28** *En los sistemas 'cloud' modernos, la elasticidad del servicio está gestionada por el nivel:*
- a** SaaS.
  - b** PaaS.
  - c** IaaS.
  - d** Cualquier nivel.

- 29** *El siguiente fragmento de 'docker-compose.yml' fue utilizado en la primera sesión de la Práctica 3 para configurar los componentes 'bro' y 'wor':*

```
version: '2'
services:
  # Clients are not relevant here.
  wor:
    image: worker
    build: ./worker/
    links:
      - bro
    environment:
      - BROKER_HOST=bro
      - BROKER_PORT=9999
  bro:
    image: broker
    build: ./broker/
    expose:
      - "9998"
      - "9999"
```

*Si no se modificaran las imágenes 'worker' y 'broker' utilizadas en esa sesión,*

*¿qué cambios deberían realizarse en el fragmento anterior para que el componente 'wor' se inicie antes que el componente 'bro' y ambos componentes sigan funcionando correctamente?*

- a** Eliminar la sección 'links' de 'wor', y añadir una sección 'links' con el valor '- wor' en 'bro'.
- b** Ninguno. El orden de inicio solicitado no puede implantarse, pues el código del worker depende de un inicio previo del broker.
- c** Trasladar las secciones 'links' y 'environment' de 'wor' a 'bro', sustituyendo todo valor 'bro' que haya en ellas por 'wor'.
- d** No hay que hacer ningún cambio, pues el orden de inicio configurado en ese fragmento ya coincide con el solicitado.



- 30** El siguiente fragmento de 'docker-compose.yml' fue utilizado en la primera sesión de la Práctica 3 para configurar el componente 'wor':

```
version: '2'
services:
# Clients and broker are not relevant here.
  wor:
    image: worker
    build: ./worker/
    links:
      - bro
    environment:
      - BROKER_HOST=bro
      - BROKER_PORT=9999
```

¿Cómo consigue el programa worker.js, utilizado para crear la imagen 'worker', obtener el valor de la variable BROKER\_HOST?

- a La sección 'environment:' de un 'docker-compose.yml' es meramente informativa y no tiene ningún efecto práctico. El programa no usa esa variable para nada.
- b El programa dispone de una variable llamada BROKER\_HOST y Docker le asignará el valor asociado a 'bro' cuando el contenedor se inicie.
- c La variable BROKER\_HOST se utilizó en el Dockerfile asociado a la imagen 'worker' y permite que worker.js obtenga su valor en alguno de sus argumentos.
- d Ninguna de las demás alternativas es cierta.

- 31** El siguiente fragmento de 'docker-compose.yml' fue utilizado en la segunda sesión de la Práctica 3 para configurar parcialmente el componente 'bro':

```
version: '2'
services:
# Clients(cli), workers(wor) logger(log) not relevant here
  bro:
    image: broker
    build: ./broker/
    links:
      - log
    expose:
      - "9998"
      - "9999"
    ports:
      - "9998:9998"
```

¿Para qué se utiliza la sección 'ports' en ese fragmento?

- a Para indicarle al administrador del sistema que el puerto 9998 es el más importante en este despliegue. Es una sección meramente informativa, al igual que expose.
- b Para que el componente 'log', cuando se despliegue en otras máquinas, pueda interactuar con el componente 'bro'.
- c Para que los workers desplegados en otras máquinas puedan interactuar con el componente 'bro'.
- d Para que el puerto 9998 del contenedor de 'bro' se corresponda con el puerto 9998 del ordenador anfitrión.

**32** *¿Por qué en el ejercicio de despliegue de WordPress, basado en Bitnami, no aparece ningún Dockerfile?*

- a** Porque WordPress se encuentra preinstalado en la distribución LINUX de las máquinas virtuales de portal.
- b** Porque al contar con dos imágenes no puede construirse un Dockerfile que las construya a la vez.
- c** Esos Dockerfile no se descargan porque el archivo docker-compose.yml provoca su ejecución en el depósito remoto.
- d** Los Dockerfile no son necesarios porque las imágenes ya han sido construidas y docker-compose.yml se basa en ellas

Rellena y entrega la siguiente hoja de respuestas. Cada cuestión posee una única respuesta correcta. No olvides cumplimentar correctamente tus datos personales.

No taches una posible respuesta incorrecta: bórrala o cúbrela con Typex

Una cuestión con más de una respuesta marcada se considera no contestada

0	0	0	0	0	0	0	0
1	1	1	1	1	1	1	1
2	2	2	2	2	2	2	2
3	3	3	3	3	3	3	3
4	4	4	4	4	4	4	4
5	5	5	5	5	5	5	5
6	6	6	6	6	6	6	6
7	7	7	7	7	7	7	7
8	8	8	8	8	8	8	8
9	9	9	9	9	9	9	9

DNI:

Apellidos:

Nombre:

PARCIAL 1

1

A

B

C

D

2

A

B

C

D

3

A

B

C

D

4

A

B

C

D

5

A

B

C

D

6

A

B

C

D

7

A

B

C

D

8

A

B

C

D

9

A

B

C

D

10

A

B

C

D

11

A

B

C

D

12

A

B

C

D

13

A

B

C

D

14

A

B

C

D

15

A

B

C

D

16

A

B

C

D

PARCIAL 2

17

A

B

C

D

18

A

B

C

D

19

A

B

C

D

20

A

B

C

D

21

A

B

C

D

22

A

B

C

D

23

A

B

C

D

24

A

B

C

D

25

A

B

C

D

26

A

B

C

D

27

A

B

C

D

28

A

B

C

D

29

A

B

C

D

30

A

B

C

D

31

A

B

C

D

32

A

B

C

D