Лекція 8
Базові комбінаторні алгоритми

План лекції

1. **Алгоритми породження підмножин**
2. **Генерування всіх підмножин**
3. **Алгоритм генерації всіх двійкових векторів довжини n в лексикографічному порядку**
4. **Генерування підмножин з умовою**
5. **Генерування k-елементних підмножин**
6. **Алгоритми перестановок**
7. **Вибір за допомогою сортування**

7.1. Швидке сортування
7.2. Сортування злиттям
7.3. Бінарний пошук елементів у масиві.

1. Алгоритми породження підмножин.

Розглянемо задачу генерування підмножин довільної n-множини

$$A = \{a_1, a_2, ..., a_n\}.$$

Існує кілька варіантів цієї задачі, які включають:

- генерування всіх можливих підмножин даної множини,

- генерування підмножин з деякими умовами, які накладають на, підмножини.

Відомо, що кожна n-множина A має точно 2^n підмножин (згадайте белеан). Тому при створенні комп'ютерних алгоритмів породження підмножин доцільно кожну з отриманих підмножин представити у вигляді двійкової послідовності.

Деяка підмножина $B \subset A$ може бути зіставлена з двійковою послідовністю $b_1 b_2 ... b_j ... b_n$, яку задають в такий спосіб:

$$b_j = \begin{cases} 0, & a_j \notin B, \\ 1, & a_j \in B. \end{cases}$$

Такий підхід дозволяє встановити взаємно однозначну відповідність між усіма булевими векторами довжини n і елементами множини B.

2. Генерування всіх підмножин

Для генерації всіх підмножин n-множини A, досить породити всі двійкові набори довжини n.

Легко побачити, що найбільш прямим способом їх породження є запис у системі числення з основою 2.

Приклад. Згенерувати всі підмножини множини $M=\{a_0,a_1,a_2\}$

Розв'язок. Позначимо $i-$у підмножину множини M через B_i, де $i=1,2,...,2^{|M|}$

Кожній підмножині B_i поставимо у відповідність двійкову послідовність $b_0b_1b_2...b_{|M|-1}$ з умовою, що

$$b_j=\begin{cases}0, & a_j\notin B,\\ 1, & a_j\in B.\end{cases}$$

Результати відповідності представимо в таблиці:

i	$b_0b_1b_2$	B_i
0	000	$\varnothing$
1	001	a_2
2	010	a_1
3	011	a_1,a_2
4	100	a_0
5	101	a_0,a_2
6	110	a_0,a_1
7	111	a_0,a_1,a_2

$$2^M=\{\varnothing,a_0,a_1,a_2,\{a_0,a_1\},\{a_0,a_2\},\{a_1,a_2\},\{a_0,a_1,a_2\}\}$$

$$2^{|M|}=3,\quad b_0...b_{|M|-1}=b_0b_1b_2$$

3. Алгоритм генерації всіх двійкових векторів довжини n в лексикографічному порядку

Алгоритм породжує всі двійкові вектори $b=(b_{n-1},b_{n-1},...,b_1,b_0)$ довжини n в лексикографічному порядку, починаючи з найменшого елемента.

1. Будемо використовувати масив `b[n],b[n-1], ….,b[1],b[0],` установивши `b[n]:=0.`

2. Переглядаючи справа наліво, знаходимо першу позицію `b[i]` таку, що `b[i]=0`.
3. Записуємо `b[i]:=1`, а всі елементи `b[j]`, `j<i`, що розміщені праворуч від `b[i]`, встановлюємо в `0`.
4. Для всіх породжуваних послідовностей елемент `b[n]` не змінюється, за винятком генерації останнього вектора `(1,1,…,1)`, `i=n`. Рівність `b[n]=1` є умовою зупинки алгоритму.

Псевдокод програми генерації двійкових векторів довжини `n`

```
For i:=0 to n do b[i]:=0;  [початковий вектор]
While b[n]≠ 1 do [перевірка закінчення алгоритму ]
begin
 Write(b[n-1], b[n-2],…, b[0]);
 i=0;
 While b[i]=1 do
 begin
  b[i]:=0;
  i:=i+1;
 end;
 b[i]:=1;
end;
```

n=3
2 1 0
0 0 0
0 0 1
0 1 0
0 1 1
1 1 1

Розглянемо генерацію підмножин множини $A=\{a_0,a_1,...,a_{n-1}\}$.
Введемо фіктивний елемент $a_n \notin A$.

Генерація всіх двійкових векторів `b` довжини $n=3$ й підмножин B множини $A=\{a_0,a_1,a_2\}$.

$b^1=(0,0,0),\ B^1=\varnothing,\ i=1;$

$b^2=(0,0,1),\ B^2=\{a_2\},\ i=2;$

$b^3=(0,1,0),\ B^3=\{a_1\},\ i=0;$

$b^4=(0,1,1),\ B^4=\{a_1,a_2\},\ i=2;$

$b^5=(1,0,0),\ B^5=\{a_0\},\ i=0;$

$b^6=(1,0,1),\ B^6=\{a_0,a_2\},\ i=1;$

$b^7=(1,1,0),\ B^7=\{a_0,a_1\},\ i=0;$

$b^8=(1,1,1),\ B^8=\{a_0,a_1,a_2\},\ i=3.$

```
 B := ∅;
 While a_n ∉ B do
 begin
  Write(B);
  i := 0;
  While a_i ∈ B
do
  begin
   B := B \ {a_i};
      i := i+1;
  end;
  B := B ∪ {a_i};
 end;
```

4. Генерування підмножин з умовою

Генерація підмножин з умовою мінімальної відмінності сусідніх породжуваних елементів.

Для такої генерації скористаємося записом чисел у двійковому *коді Грея.*

Алгоритм одержання коду Грея.

Нехай $b_1 b_2 ... b_n$ – деяке двійкове число.

Перший спосіб генерації коду Грея. Код Грея числа одержують шляхом зсуву його на один розряд вправо, і, відкинувши самий правий (n-й розряд), додають порозрядно по модулю два із цим же, але незсунутим числом:

$$\begin{array}{l} b_1 b_2 b_3 ... b_{n-1} b_n \\ \oplus\, b_1 b_2 b_3 ...... b_{n-1} \cancel{b_n} \\ \hline c_1 c_2 c_3 .... c_{n-1} c_n \end{array}$$

Правило додавання

$b_i \oplus b_{i-1}$	c_i
$0 \oplus 0$	0
$0 \oplus 1$	1
$1 \oplus 0$	1
$1 \oplus 1$	0

Таким чином, кожний результуючий розряд c_i,. одержують по формулі $c_i = b_i \oplus b_{i-1}$. Вважають, що $b_0 = 0$.

Приклад. Розглянемо порядок генерації коду Грея для трьохрозрядних двійкових чисел:

i	Двійкове число	Операція	Код Грея
0	000	$000 \oplus 00 = 000$	000
1	001	$001 \oplus 00 = 001$	001
2	010	$010 \oplus 01 = 011$	011
3	011	$011 \oplus 01 = 010$	010
4	100	$100 \oplus 10 = 110$	110
5	101	$101 \oplus 10 = 111$	111
6	110	$110 \oplus 11 = 101$	101
7	111	$111 \oplus 11 = 100$	100

Другий спосіб генерації коду Грея

Другий спосіб генерації коду Грея складається з наступних кроків:

1. У якості базової використовуємо дворозрядну послідовність: 00,01,11,10.
2. Будуємо трьохрозрядну послідовність:

2а. Припишемо до 00,01,11,10 праворуч 0:

000,010,110,100.

2б. Переставимо елементи 00,01,11,10 у зворотному порядку:

10,11,01,00.

2в. До елементів 10,11,01,00 припишемо праворуч 1:

101,111,011,001.

2г. Об'єднаємо послідовності з п.2а й п.2в:

000, 010,110,100,101,111,011,001.

3. Для одержання чотирьохрозрядної послідовності перейдемо до п.1 алгоритму, замінивши дворозрядну послідовність послідовністю, отриманої в п. 2г.
4. Повторюючи дії $n-2$ разу, одержимо $n-$розрядний код Грея.

Правило генерації коду Грея другим способом

Якщо послідовність $c_1, c_2, c_3, ..., c_k$ містить усі двійкові послідовності довжини k й кожний член послідовності відрізняється від сусіднього точно одним елементом, то, приписуючи праворуч нуль до кожного члена цієї послідовності і одиницю також до кожного члена цієї послідовності, записаної у зворотному порядку, одержимо в результаті нову послідовність, яка містить усі послідовності довжини $k+1$ з сусідніми членами, які відрізняються один від одного точно однією координатою.

Приклад. Згенерувати всі підмножини множини $A = \{a_1, a_2, a_3\}$ з умовою мінімальної відмінності сусідніх породжуваних елементів.

Розв'язок. Результати представимо у вигляді таблиці:

i	$b_1 b_2 b_3$	B_i
0	000	$\varnothing$
1	010	a_2
2	110	a_1, a_2
3	100	a_1
4	101	a_1, a_3
5	111	a_1, a_2, a_3
6	011	a_2, a_3
7	001	a_3

1. Початкова послідовність

00,01,11,10

2а. Дописали 0 справа

000,010,110,100

2б. Послідовність (1) перевернули

10,11,01,00

2в. До послідовності (2б) дописали 1

101,111,011,001

2г. Поєднали послідовності (2а) та (2в)

000, 010,110,100,101,111,011,001

Псевдокод програми, що генерує всі підмножини за допомогою коду Грея першим способом

```
 Program Gray;
Var
 i,M,N:byte;
 { N-розряднісь, M=2^N-Кількість комбінацій}
  G:array[1..M] of byte;

 function Bintogray(b:byte):byte;
 begin
  Bintogray:=b xor (b shr 1)
 end;

begin (* головна програма *)
  For i:=1 to M do G[i]:=Bintogray(i);
end; (*кінець програми*)
```

5. Генерування k-елементних підмножин $n-$множини. Алгоритм генерування сполук з n по k в лексикографічному порядку

Нехай $X=\{1,2,...,n\}$. *Наприклад*: $X=\{1,2,3,4,5,6\}$, $n:=6,\ k:=4$

1. Перша послідовність $(a_1,a_2,...,a_{k-1},a_k)$. *Наприклад*: $(1,2,3,4)$

2. Якщо $(a_1<a_n-k+1)\&...\&(a_{k-1}<a_n-1)\&(a_k<a_n)$ то $(a_1,a_2,...,a_{k-1},a'_k:=a_k+1)$; goto 2.

Наприклад: якщо $(1,2,3,4)$ то $(1,2,3,5)$

якщо $(a_1<a_n-k+1)\&...\&(a_{k-1}<a_n-1)\&(a_k=a_n)$ то $(a_1,a_2,...,a'_{k-1}:=a_{k-1}+1,a'_k:=a'_{k-1}+1)$; goto2

Наприклад: якщо $(1,2,3,6)$ то $(1,2,4,5)$

……………………………………………………………………………………

3. Якщо $(a_1<a_n-k+1)\&...\&(a_{k-1}=a_n-1)\&(a_k=a_n)$ то $(a'_1:=a_1+1,a'_2:=a'_1+1,...,a'_{k-1}:=a'_1+k-2,a'_k:=a'_1+k-1)$; goto2;

Наприклад: якщо $(1,4,5,6)$ то $(2,3,4,5)$

4. Якщо $(a_1=a_n-k+1)\&...\&(a_{k-1}=a_n-1)\&(a_k=a_n)$ то Stop

Наприклад: якщо $(3,4,5,6)$ то Stop

Псевдокод алгоритму що генерує всі $k-$елементні підмножини $n-$множини

```
begin
  For i:=0 to k do A[i]:=i;
  p:=k;
  while p≥1 do
  begin
   write (A[1],…,A[k]);
   if A[k]=n then p:=p-1
   else p:=k;
   If p≥1 then
   For i:=k downto p do
   A[i]:=A[p]+i-p+1;
end;
```

1234
1235
1236
1245
1246
1256
1345
1346
1356
1456
2345
2346
2356
2456
3456

Праворуч наведений приклад 4-елементних підмножин множини {1,…,6}, побудованих по даному алгоритму.

6. Алгоритми перестановок

Розглянемо методи генерування послідовностей $n!$ перестановок множини, складеної з n елементів. Для цього задану множину представимо у вигляді елементів масиву $P[1], P[2], ..., P[n]$.

Методи, які будуть нами розглядатися, базуються на застосуванні до масиву $P[i]$, $i = 1, 2, ..., n$ елементарної операції, яку називають **поелементною транспозицією**. Суть операції полягає в обміні даними між елементами масиву $P[i]$ і $P[j]$, $1 \le i, j \le n$ за такою схемою:

$$vrem := P[i],\ P[i] := P[j],\ P[j] := vrem,$$

де $vrem -$ деяка допоміжна змінна, яку використовують для тимчасового зберігання значення елемента масиву $P[i]$.

Лексикографічний порядок

Визначення лексикографічного порядку. Нехай існують перестановки у вигляді послідовностей $\{x_1, x_2, x_3, ..., x_n\}$, $\{y_1, y_2, y_3, ..., y_n\}$,... тієї самої множини X. Перестановки з елементів множини X впорядковані в лексикографічному порядку, якщо $\{x_1, x_2, x_3, ..., x_n\} < \{y_1, y_2, y_3, ..., y_n\}$ тоді й тільки тоді, коли для довільного k: $x_k \le y_k$ і $x_i = y_i$ для всіх $i < k$.

Визначення антилексикографічного порядку. Нехай існують перестановки у вигляді послідовностей $\{x_1, x_2, x_3, ..., x_n\}$, $\{y_1, y_2, y_3, ..., y_n\}, ...$ тієї самої множини X. Перестановки з елементів множини X впорядковані в антилексикографічному порядку, якщо $\{x_1, x_2, x_3, ..., x_n\} > \{y_1, y_2, y_3, ..., y_n\}$ тоді й тільки тоді, коли для довільного k : $x_k > y_k$ і $x_i = y_i$ для всіх $i < k$.

Алгоритм побудови перестановок у лексикографічному порядку

Початкова перестановка $(1, 2, ..., n-1, n)$.

Завершальна перестановка $(n, n-1, ..., 2, 1)$.

Розглянемо перехід від $(x_1, x_2, ..., x_n)$ до $(y_1, y_2, ..., y_n)$

Як будуємо перестановку $(y_1, y_2, ..., y_n)$?

1. Розглядаємо перестановку справа наліво

$$\leftarrow$$

$$x = (x_1, x_2, ..., x_i, x_{i+1} ..., x_n)$$

і знайдемо таку позицію i, що $x_i < x_{i+1}$.

2. Якщо такої позиції немає, то тоді $x_1 > x_2 > ... > x_n$, тобто $x = (n, n-1, ..., 1)$. Дана перестановка є завершальною перестановкою нашого алгоритму.

3. Якщо позиція i знайдена, то $x_i < x_{i+1} > x_{i+2} > ... > x_n$.

4. Шукаємо першу позицію j в діапазоні від позиції n до позиції i таку, що $x_i < x_j$. Тоді $i < j$.

$$\leftarrow$$

$$x = (x_1, x_2, ..., x_i, x_{i+1}, ..., x_j, ..., x_n)$$

5. Далі виконуємо операцію транспозиції над елементами x_i й x_j

$$\leftarrow$$

$$x = (x_1, x_2, ..., x_i, x_{i+1}, ..., x_j, ..., x_n)$$

6. В отриманій перестановці частину елементів $x_{i+1}, ..., x_{n-1}, x_n$ перевертаємо, тобто міняємо порядок їх слідування на протилежний.

7. Отримана перестановка є перестановкою $y = (y_1, y_2, ..., y_n)$.

Саме ця перестановка є наступною в лексикографічному порядку слідування перестановок.

Приклад. Нехай $x = (2, 6, 5, 8, 7, 4, 3, 1)$.

1. Відповідно до даного алгоритму, $x_i = 5$, а $x_j = 7$.

2. Виконаємо транспозицію для елементів $i = 3$ і $j = 5$:

$$\tilde{x} = (2,6,7,8,5,4,3,1)$$

3. Перевернемо частину елементів $x_3,...,x_8 \to x_8,...,x_3$:

$$(8,5,4,3,1) \to (1,3,4,5,8).$$

У результаті одержимо наступну в лексикографічному порядку перестановку $y = (2,6,7,1,3,4,5,8)$

Псевдокод програми генерування перестановок у лексикографічному порядку

Елемент масиву a[0]=0 використовується тільки як ознака закінчення алгоритму.

```
For j:=0 to n do a[j]:=j;{генерація поч. посл.}
i:=1;
while i≠ 0 do
begin
 write(a[1],a[2],…,a[n]);
 i:=n-1;                              {пошук a[i]}
 while a[i]>a[i+1] do i:=i-1;
 j:=n;                                {пошук a[j]}
 while a[j]<a[i] do j:=j-1;
 Swap(a[i],a[j]);

 {перевертання частини послідовності}
 k:=i+1;
 m:=i+tranc((n-1)/2) ;
 while k≤ m do
 begin
  Swap(a[k],a[n-k+i+1]);
  k:=k+1;
 end;
end;
```

Приклад. При $n=3$ процес роботи даного алгоритму представлений наступною послідовністю перебудувань перестановок a^k.

```
a¹={123},   a¹[i]=2,   a¹[j]=3;
a²={132},   a²[i]=1,   a²[j]=2;
a³={213},   a³[i]=1,   a³[j]=3;
a⁴={231},   a⁴[i]=1,   a⁴[j]=3;
a⁵={312},   a⁵[i]=1,   a⁵[j]=2;
a⁶={321},   i=0;
```

Перестановки множин $X=\{1,2,3\}$ у лесикографічному (а) і антилексикографічному (б) порядку

	(а)	(б)
1	1 2 3	1 2 3
2	1 3 2	2 1 3
3	2 1 3	1 3 2
4	2 3 1	3 1 2
5	3 1 2	2 3 1
6	3 2 1	3 2 1

Словниковий порядок

Приклад. *Розбивка числа 7 у словниковому порядку.*

$(7\cdot 1) = (1, 1, 1, 1, 1, 1, 1)$,
$(1\cdot 2, 5\cdot 1) = (2, 1, 1, 1, 1, 1)$,
$(2\cdot 2, 3\cdot 1) = (2, 2, 1, 1, 1)$,
$(3\cdot 2, 1\cdot 1) = (2, 2, 2, 1)$,
$(1\cdot 3, 4\cdot 1) = (3, 1, 1, 1, 1)$,
$(1\cdot 3, 1\cdot 2, 2\cdot 1) = (3, 2, 1, 1)$,
$(1\cdot 3, 2\cdot 2) = (3, 2, 2)$,
$(2\cdot 3, 1\cdot 1) = (3, 3, 1)$,
$(1\cdot 4, 3\cdot 1) = (4, 1, 1, 1)$,
$(1\cdot 4, 1\cdot 2, 1\cdot 1) = (4, 2, 1)$,
$(1\cdot 4, 1\cdot 3) = (4, 3)$,
$(1\cdot 5, 2\cdot 1) = (5, 1, 1)$,
$(1\cdot 5, 1\cdot 2) = (5, 2)$,
$(1\cdot 6, 1\cdot 1) = (6, 1)$,

7. Вибір за допомогою сортування

Задачу вибору можна звести до СОРТУВАННЯ.

Як це зробити?

1. Упорядкувати масив. Обчислювальна складність O(*n* log *n*)

2. Взяти потрібний по порядку елемент.

Коли це вигідно застосовувати?

Це ефективно в тому випадку, коли вибір потрібно робити багаторазово

При однократному виборі алгоритм не ефективний

7.1. «Швидке сортування» (Quicksort).

У наведеному алгоритмі використані наступні позначення:

a[k] - масив чисел, у якім проводиться сортування.

Процедура дозволяє сортувати довільна підмножина масиву a[k]

g – номер мінімального елемента масиву, з якого починається сортування.

r – номер максимального елемента масиву, на якім закінчується сортування.

Суть методу

1. На кожній ітерації виділяють частину масиву чисел — робочий масив.

На першій ітерації – це весь масив чисел a[k], у якому проводиться сортування. Встановлюємо границі робочого масиву $g=1$ і $r=n$. 2. Вибирають елемент x:=a[(g+r)div2], який розміщений посередині робочого масиву.

3. Далі, починаючи з $i=1$, послідовно збільшуємо значення i на одиницю й порівнюємо кожний елемент a_i з x, поки не буде знайдено елемент a_i такий, що $a_i > x$.

4. Потім, починаючи з $j=r$, послідовно зменшуємо значення j на одиницю, поки не буде знайдений елемент a_j такий, що $x > a_j$.

5. Якщо для знайдених елементів a_i і a_j виконується умова $i \le j$, то ці елементи в робочому масиві міняються місцями.

6. Описана процедура закінчиться знаходженням головного елемента й поділом масиву на дві частини: одна частина буде містити елементи, менші x, а інша — більші x.

$\leqslant x$	x	$x \geqslant$

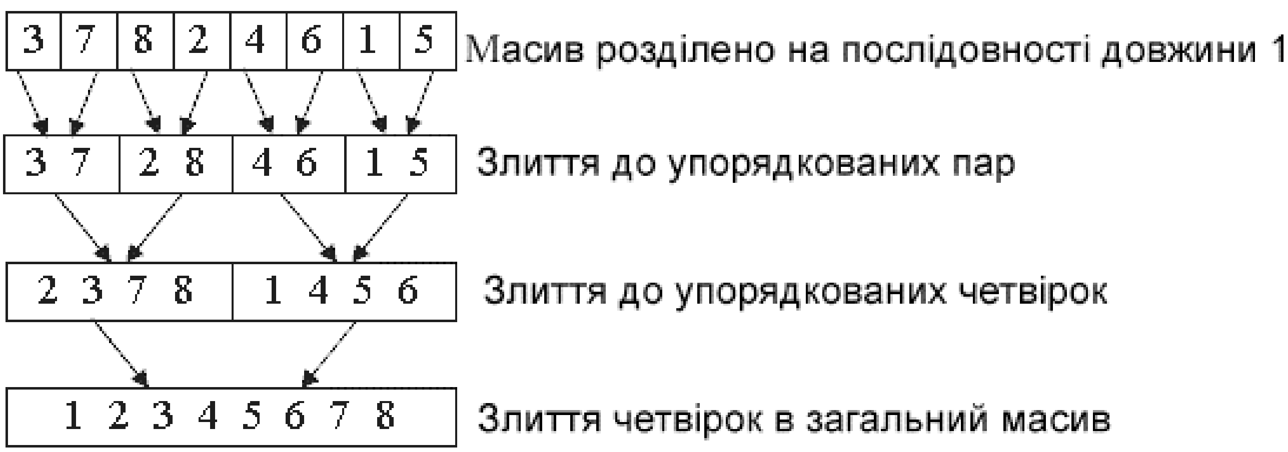
Далі для кожної такої частини знову застосовується описана вище процедура. Паскаль-Програма, що реалізує даний алгоритм для 10-елементного масиву, має такий вигляд:

```
Program Qsort;
Const N=10;
Var a:array[1..N] of integer; (* вихідний масив *)
     k:integer;
procedure Quicksort(g,r:integer);
(* Процедура швидкого сортування*)
var i,j,x,y: integer;
begin
 i:=g; j:=r; x:= a[(g+r) div 2];
 repeat
  while (a[i]<x) do inc(i);
  while (x<a[j]) do dec(j);
  if (i<=j) then
  begin
   y:=a[i]; a[i]:=a[j]; a[j]:=y; inc(i); dec(j);
  end;
 until (i>j);
(*Рекурсивне використання процедури Quicksort *)
if (g<j) then Quicksort(g,j);
if (i<r) then Quicksort(i,r);
end;
begin
  writeln('Уведіть',N, 'елементів масиву:') ;
  for k:=1 to N do readln(a[k]);
  Quicksort(1,N);
  writeln('Після сортування:');
  for до:=1 to N do write(a[k],' ');
end.
```

7.2.Сортування злиттям

Сортування злиттям також побудована на принципі " розділяй-і-пануй", однак реалізує його трохи по-іншому, ніж Quick Sort. А саме, замість поділу по опорному елементу масив просто ділиться навпіл.

Функція Merge на місці двох упорядкованих масивів A[left]...A[mid] і A[mid+1]...A[right] створює єдиний упорядкований масив A[left]...A[right]. Приклад роботи алгоритму на масиві 3 7 8 2 4 6 1 5..



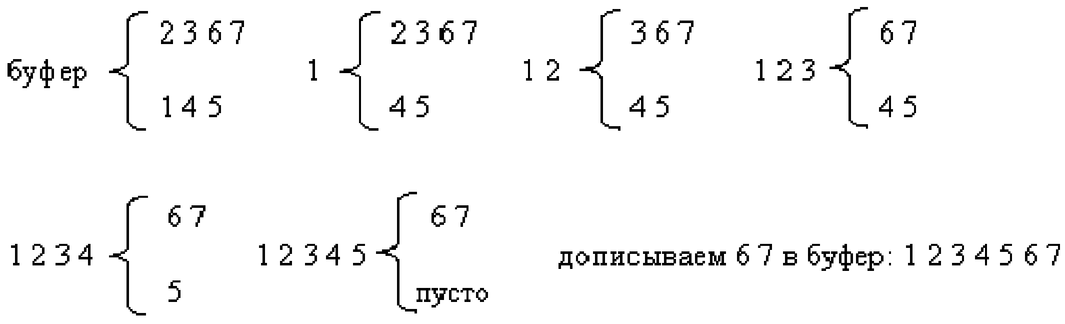
```
Program Mrgesort;
Var A,B : array[1..1000] of integer;
    N  : integer;

Procedure Merge(left,right : integer); {процедура, що
зливає масиви}
Var mid,i,j,k : integer;
Begin
 mid:=(left+right) div 2;
 i:=left;
 j:=mid+1;
 for k:=left to right do
 if (i<=mid) and ((j>right) or (A[i]<A[j])) then
 begin
   B[k]:=A[i];
   i:=i+1;
 end  else
 begin
   B[k]:=A[j];
   j:=j+1;
 end ;
 for k:=left to right do A[k]:=B[k];
End;




{left,right - індекси початку й кінця частини масиву, яку
сортують}

Procedure Sort(left,right : integer);
Begin
 {масив з одного елемента тривіально впорядкований}
 if left<right then
 begin
  mid:=(left+right) div 2
  Sort(left,mid);
  Sort((mid + 1,right);
  Merge(left,right);
 end;
End;
Begin
 {Визначення розміру масиву A(N) і його заповнення}
```

```
 …
 {запуск процедури, яка сортує, }
 Sort(1,N);
 {Вивід відсортованого масиву A}
  …
End.
```

Один зі способів полягає в злитті двох упорядкованих послідовностей за допомогою допоміжного буфера, рівного по розміру загальній кількості наявних у них елементів. Елементи послідовностей будуть переміщатися в цей буфер по одному за крок.

```
merge ( упорядковані послідовності A, B, буфер C ) {
поки A і B непорожні {зрівняти перші елементи A і B
           перемістити найменший у буфер
   }
   якщо в одній з послідовностей ще є елементи
   дописати їх у кінець буфера, зберігаючи наявний
порядок
   }
```

Приклад роботи на послідовностях 2 3 6 7 і 1 4 5



}

Оцінимо швидкодію алгоритму: час роботи визначається рекурсивною формулою T(n) = 2T(n/2) + Theta(n).

Її розв'язок: T(n) = n log n - результат досить непоганий, враховуючи відсутність "гіршого випадку". Однак, незважаючи на гарну загальну швидкодію, у сортування злиттям є й серйозний мінус: вона вимагає Theta(n) пам'яті.

Добре запрограмована внутрішнє сортування злиттям працює небагато швидше пірамідальної, але повільніше швидкої, при цьому вимагаючи багато пам'яті під буфер. Тому Mergesort використовують для впорядкування масивів, лише якщо потрібна стійкість методу(якої немає ні у швидкої, ні в пірамідальної сортувань).

Сортування злиттям є одним з найбільш ефективних методів для однозв'язних списків і файлів, коли є лише послідовний доступ до елементів.

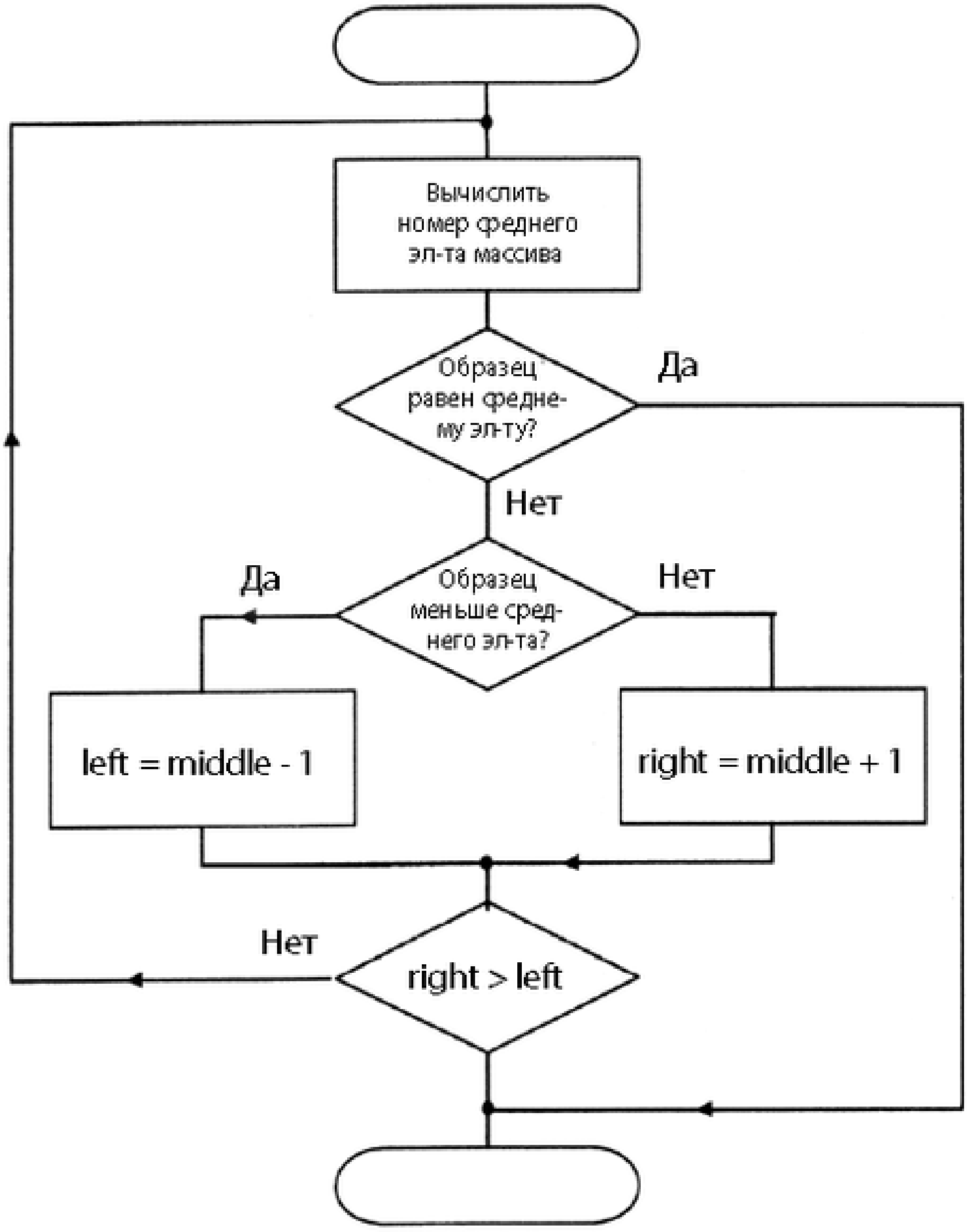
7.3. Двійковий (бінарний) пошук елемента в масиві

Якщо в нас є масив, що містить упорядковану послідовність даних, то дуже ефективний двійковий пошук.

Ідея пошуку полягає в тому, щоб брати елемент посередине, між границями, і порівнювати його із шуканим. У випадку рівності повертати його, а якщо шукане більше(у випадку правобічного - не менше), чому елемент порівняння, те звужуємо область пошуку так, щоб нова ліва границя була дорівнює індексу середини попередньої області. А якщо ні, то привласнюємо це значення правій границі. Проробляємо цю процедуру доти, поки права границя більше лівої більш ніж на 1, або ж поки ми не знайдемо шуканий індекс.

Двійковий пошук - дуже потужний метод. Якщо, наприклад, довжина масиву рівна 1023, після першого порівняння область звужується до 511 елементів, а після другий - до 255. Легко порахувати, що для пошуку в масиві з 1023 елементів досить 10 порівнянь.

.



Вычислить
номер феднего
эл-та массива
Образец
равен федне-
му эл-ту?
Да
Нет
Образец
меньше сред-
него эл-та?
Да
Нет
left = middle - 1
right = middle + 1
Нет
right > left