BinarySearch (A[1..n] : array of numbers, x : number):

в тях сме задефинирали „=“ numbers

инварианта

```
i ← 1
j ← n
while i ≤ j do
    k ← ⌊(i+j)/2⌋
    if x = A[k]
        return k
    else if x < A[k]
        j ← k-1
    else
        i ← k+1
return -1
```

Като precondition предполагаме, че масива A[1..n] е сортиран, защото binary search ще работи коректно (предстои да го покажем), ако входния масив е сортиран спрямо релацията „≤“ (казвам го така, защото в общия случай може да имате произволен тип данни, но стига спрямо релацията примерно R, да са линейно наредими (т.е. в редичка да ги подредим кой е I^ви, II^ри итн.) и операциите „=" и „R" да са ефективни (т.е. да са изчислими с компютър), то ние може в/у тях да извършим binary search при условие, че сме ги сортирали предварително като му ги подаваме.

Сега за инвариантата. В час избързах и ни трябват още две условия (един ваш колега загатна да го каже едното, но не го взех под внимание).

Инварианта: На всяко достигане на ред 3) и по точно проверката „$i<j$" е в сила:

1) $j-i \le \frac{n}{2^s}$, s - # влизания досега в цикъла

Т.е. 1) когато разстоянието м/у i и j се скъсява всеки път двойно (помага за смятане на сложност)

2) $x \in A[i \dots j]$

Реално това ни е предположението от още при стартирането на алгоритъма,

Сега да видим тези условия дали са ни достатъчни:

Индукция по $\underline{s}$:

<u>База</u>: за $s=0$ (нито веднъж не е изпълнен while цикъла), тогава $i=1$, $j=n$ и

1) $j-i = n-1 \leq n = \frac{n}{2^0}$

2) $x \in A[1 \dots n]$ да по предположение при стартиране на алгоритъма

(ИП) Допускаме, че за някое валидно s, което не е последно, то инвариантата е вярна

(Стъпка): Преминали сме през текущата итерация на while и съответно сме променили ст-тите на променливите, от които зависи инвариантата и отново сме на ред 3) преди проверката „$i < j$".
Трябва да видим, че е вярна инвариантата т.е.:

1) $j_{new} - i_{new} \leq \frac{n}{2^{s_{new}}}$ $s_{new} = s + 1$

2) $x \in A[i_{new} \dots j_{new}]$

Проверяваме ред по ред какво е могло да се промени?

4) $k \leftarrow \left\lfloor \frac{i+j}{2} \right\rfloor$

Сл. 1 $x = A[k]$

Тогава $x \in \{A[i], A[i+1], \dots, \underbrace{A\left[\left\lfloor \frac{i+j}{2} \right\rfloor\right]}_{= A[k]}, \dots, A[j]\}$

и алгоритъмът приключва с
return k

Сл. 2 $x < A[k]$

Тогава $j_{new} = k-1$, $i_{new} = i$

$$j_{new} - i_{new} = \left\lfloor \frac{i+j}{2} \right\rfloor - 1 - i \le \frac{i+j}{2} - i =$$

$$= \frac{j-i}{2} \underset{\substack{\uparrow \\ (\text{и.п})}}{\le} \frac{n}{2 \cdot 2^s} = \frac{n}{2^{s+1}} = \frac{n}{2^{s_{new}}}$$

(използвам precondition $A[1..n]$ е сортиран)

Т.к. $x < A[k]$, то знаем
$x \notin A[k, \dots, j]$ и знаем + (и.п)
$\underline{x \in A[i, \dots, (k-1)]}$, а $j_{new} = k-1$ т.е.
$x \in A[i_{new}, \dots, j_{new}]$

Сл. 3 $A[k] < x$

Тогава $i_{new} = k+1$, $j_{new} = j$

$$j_{new} - i_{new} = j - \left\lfloor \frac{i+j}{2} \right\rfloor - 1 \le j - \frac{i+j}{2} =$$

$$= \frac{j-i}{2} \underset{\substack{\uparrow \\ \text{ип}}}{\le} \frac{n}{2 \cdot 2^s} = \frac{n}{2^{s+1}} = \frac{n}{2^{s_{new}}}$$

т.к. $x > A[k]$ (използваме precondition $A[1..n]$ е сортиран)

то $x \notin A[i \ldots k]$ и $\notin (k..n)$, което беше $x \in A[i \ldots j]$, то
$x \in [(k+1) \ldots j]$, но $i_{new} = k+1$,
$j_{new} = j$

и търсим $x \in [i_{new} \ldots j_{new}]$

Добре, сега да видим дали терминира този алгоритъм и, ако терминира, той връща ли коректен резултат.

Т.к. $\{j_t - i_t\}_{t \in \mathbb{N}}$ е монотонно (строго) намаляваща редица, т.е.
спрямо това на коя итерация сме

$0 \le j_t - i_t \le \frac{n}{2^t}$, то за някое

условието за while цикъла

$t \in \mathbb{N} \to j_t - i_t < 0$ т.е. $i_t > j_t$ и
$\exists m \in \mathbb{N}: i_t = j_t + m$

Т.е. while цикълът не е безкраен. Всеки път скъсяваме разстоянието м/у тях докато се разминат.

Как да дадем някаква оценка на това колко пъти се е изпълнил.

Ами имаме, че на всяка стъпка, която не е последна е в сила следното: $1 \leq j - i \leq \frac{n}{2^{(s-1)}}$

s-1 е, защото при $i=j$, то $j-i=0$ и не важи неравенството!

(това е специален случай $i=j$ - въпреки се случва, но трябва да го отчетем)

Тогава $1 \leq \frac{n}{2^{s-1}} \Leftrightarrow$

$n \geq 2^{s-1}$ // lg

$\lg n \geq s-1$

т.е. $\lg n - 1 \geq \underline{s}$ — влизания в while цикъла (най-много $\lg n - 1$ пъти се влиза)

Значи $\Theta(\lg n)$.

Добре, а коректност, че връща правилен резултат?

Вече видяхме, че терминира, така че да разгледаме, когато терминира, как се държи алгоритъма.

Нашите очаквания са следните:

хипотеза

По запускане сме предположили, че $x \in A[1..n]$. Искаме алгоритъмът следното да направи:

Ако хипотезата е вярна, то да върне $k \in \{1, ..., n\}$ – индекса на елемента x в масива $A[1..n]$ (като precondition е сортиран).

Ако хипотезата не е вярна, то да върне -1. Защо -1? Защото индексите на масива са от 1 до положителни естествени числа, а -1 е отрицателно цяло.

Случаи, в които завършва алго.

(Сл. 1) За някоя итерация t на while цикъла имаме, че $x = A[k_t]$, където

От (инв) $x \in A[i_t \ldots j_t]$ $k_t = \left\lfloor \frac{i_t + j_t}{2} \right\rfloor$

Тогава прави return k_t и т.к. k_t реално се изразява чрез i_t и j_t, то имаме $i_t \le k_t \le j_t$. Сега виждаме, че към инварианта ни ни трябва още едно условие и то е, че $1 \le i_t$ и $j_t \le n$. Лесно се доказва, защото i_t го увеличаваме само (не строго, на някои итерации не го променяме), а j_t го намаляваме (отново не строго), т.е. никога не намаляме i_t и не увеличаваме j_t, за да го счупим. Така имаме $1 \le i_t \le k_t \le j_t \le n$ т.е. $1 \le k_t \le n$, което искахме и да покажем.

Остава случаят, когато се е счупило

условието за while т.е.
за някое $t \in \mathbb{N}$, то $j_t - i_t < 0$ т.е
$j_t < i_t$. Това от инварианта
имаме, че

$$x \in A[i_t \ldots j_t] = A[\,]$$

празен масив

т.е. x не е вътре в масива
$A[1 \ldots n]$ и на последния
ред връщам return -1, което
ни показва коректно, че
$x \notin A[1 \ldots n]$.

Не сметнах за добре в час, че
е толкова дълго доказателството
и с вратички, това че се
извинявам. Ако имате въпроси,
пишете ми.

Смятане на сложности на фрагменти:

прим ① |

```
a ← 0                          const
for i ← 1 to n do
    for j ← 1 to i do
        a ← a + 1              const
return a                       const
```

$1+2+3+\dots+n = n+(n-1)+(n-2)+\dots+2+1 =$

стъпки на циклите

$= (n+1).\frac{n}{2} \approx n^2$

прим ② |

```
a ← 0
for i ← 1 to n do
    for j ← 1 to n do
        for k ← n+i+j-3 to n do
            a ← a + 1          const
return a                       const
```

за $i=1, j=1$ — $k=n-1$ — влиза

за $i=1, j=2$ — $k=n$ — влиза

за $i=2, j=1$ — $k=n$ — влиза

Точно 3 пъти влизаме в този цикъл, т.е сложността му е const.

Така цялото има сложност $\Theta(n^2)$

пример 3) Имаме масив $A = [1, 2, \dots, n]$, стек S,

предикат $P(S)$ – сложност $\Theta(1)$

↳ true, ако в S има поне 2 елем.

↳ false, иначе

```
push(A[1], S)
push(A[2], S)
for i ← 3 to n do
    while P(S) do
        pop(S)
    push(A[i], S)
```

инварианта: всеки път на достигане на проверката за влизане в цикъла в S има поне 2 елемента

Сложност $T(n) = \Theta(n)$

На страница 13 от 15 на файла, който съм качила допълнително е следващата задача, която изровихме с древния алгоритъм за умножение на две числа посредством стъпки и етикети и смятам, че и от детайли към история, и обяснения е по изчерпателно от това, което аз бих описала.

```
alg (A[1,..,n]: array of integers)
    s ← 0
    i ← 1
    j ← n
    while i < j do
        s ← s + A[i] + A[j]
        i ← i + 1
        j ← j - 1
    if n is odd
        s ← s + A[(n+1)/2]
    return s
```

$0 < j - i$

Събира всичките елем. на масива за n-четно

условие

добавя за n odd

средния елемент на масива

Тв! Alg връща сумата на всички числа в масива $A[1..n]$

Инв: При всяко достигане на проверката $i<j$:

$$s = \sum_{k=1}^{(i-1)} A[k] + \sum_{k=(j+1)}^{n} A[k] \quad \text{и} \quad i-1 = n-j$$

Д-во инварианта:

База: $s=0$, $i=1$, $j=n$: $S = 0 = \sum_{k=1}^{0} A[k] + \sum_{k=n+1}^{n} A[k] = 0+0$.

и $1-1 = n-n \Leftrightarrow 0=0$

(И.П.) Нека е в сила за някое достигане, което не е последно.

След ред 5) $s_{new} = s + A[i] + A[j] \overset{И.П.}{=}$

$$= \sum_{k=1}^{(i-1)} A[k] + \sum_{k=j+1}^{n} A[k] + A[i] + A[j] =$$

$$= \sum_{k=1}^{i} A[k] + \sum_{k=j}^{n} A[k]$$

$i_{new} = i+1$, $j_{new} = j-1$ и спрямо тях

$$s_{new} = \sum_{k=1}^{i_{new}-1} A[k] + \sum_{k=j_{new}+1}^{n} A[k]$$, а

$$i_{new} - 1 \underset{ИП}{=} i+1-1 = (i-1)+1 \underset{ИП}{=} (n-j)+1$$

$$n - j_{new} \underset{ИП}{=} n-(j-1) = (n-j)+1$$

Терминация:

За последно достигане знаем $\underline{i \geq j}$

(1сл) n е нечетно

Тогава $i = j$.

От инв: $\underline{i-1 = n-j}$ т.е. $i-1 = n-i \Leftrightarrow$

$\underline{n = 2i+1}$

и $s = \sum_{k=1}^{i-1} A[k] + \sum_{k=i+1}^{n} A[k]$ и с if-а става $s = \sum_{k=1}^{n} A[k]$

(Сл2) n е четно

Тогава $i = j+1$

От инв: $i-1 = n-j \Leftrightarrow j+1-1 = n-j \Leftrightarrow$

$n = 2j$ и $s = \sum_{k=1}^{j} A[k] + \sum_{k=j}^{n} A[k] = \sum_{k=1}^{n} A[k]$.

Другият път ще направим още задачи за доказателство на коректност на итеративни алгоритми и смятане на сложност. Затова мисля да си е 3 часа, за да си наваксаме от 2те седмици пропуск. Сега ще сложа фрагменти на два от алгоритмите, чиято коректност искам да докажем другия път и си помислете за инварианти:

Зад / Kadane: максимална сума на подмасив

```
kadane(A[1..n]: array of numbers): number
  localMax ← 0
  globalMax ← -∞
  for i ← 1 to n do              ← инварианта?
      localMax ← max(A[i], A[i] + localMax)
      globalMax ← max(localMax, globalMax)
  return globalMax
```

Зад / Selection sort

Докажете коректност и изведете скоростта на алгоритъма.

```
sort(A[1...n]: array of numbers): void
  for i ← 1 to n-1 do            ← инв 1
      min ← i
      for j ← i+1 to n do        ← инв 2
          if A[j] < A[min] then
              min ← j
      swap(A[i], A[j])
```

Още една задача от контролно.
Помислете какво връща и каква асимптотична сложност има?

```
alg(A[1...n] : array of numbers): number
i ← 1
while A[i] ≤ 0 and i ≤ n do
        i ← i+1
if i = n+1
        return EmptySet
j ← i
temp ← A[i]
max ← 0
while j ≤ n
        if temp > max
                max ← temp
                maxi ← i
                maxj ← j
        j ← j+1
        temp ← temp + A[j]
        if temp ≤ 0
                temp ← 0
                i ← j+1
return (maxi, maxj)
```